Supplementary Information for:

ShinyCell: Simple and sharable visualisation of single-cell gene expression data

John F. Ouyang, Uma S. Kamaraj, Elaine Y. Cao and Owen J. L. Rackham*

Availability: ShinyCell is available at https://github.com/SGDDNB/ShinyCell.
Contact: owen.rackham@duke-nus.edu.sg

ShinyCell is an R package that converts scRNA-seq datasets into explorable and shareable interactive interfaces. Here, we provide supplementary information, including comparison of ShinyCell with similar packages and compiled READMEs / tutorials on how to use ShinyCell.

S1. Comparison of ShinyCell with similar packages

S1.1 scRNA-seq visualisation tools to be compared

To highlight and compare the features of the ShinyCell package, we identified a list of popular scRNA-seq visualisation tools, focussing on tools that allow for web sharing. This is because a key objective of ShinyCell is to enable the online sharing of complex scRNA-seq dataset to non-computational researchers. Thus, we considered nine scRNA-seq visualisation tools: ASAP[1], cellxgene[2], iSEE[3], Loom viewer[4], SCope[5], scSVA[6], Single Cell Explorer[7], SPRING[8] and UCSC Cell Browser[9].

We then make a further distinction between tools that are mainly designed for visualisation and tools that incorporate both single-cell preprocessing/analysis and visualisation. The tools ASAP, scSVA and SPRING fall into the latter category. All tools will be compared with ShinyCell in terms of (i) the key functionalities, (ii) the display / UI options and (iii) the input file formats that are supported (Table 1).

Table 1: Comparison of different scRNA-seq visualisation tools in terms of key functionalities (orange rows), display / UI options (blue rows), supported input file formats (green rows). These comparisons will be explained in greater detail in the subsections below. *Side-by-side display of low-dimensional embeddings are not enabled by default; additional panels/tabs have to be created to simulate side-by-side embeddings. **ASAP only supports the display of low-dimensional embeddings calculated by the program itself. Abbreviations: SC analysis (single-cell analysis), SaaS (Software as a Service), PDF (Portable Document Format) and PNG (Portable Network Graphics). ***Via a tutorial/code in github repository.

	ShinyCell	cellxgene	iSEE	Loom viewer	SCope	Single Cell Explorer	UCSC Cell Browser	ASAP	scSVA	SPRING
Web Sharing	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC analysis								✓	✓	✓
Docker		✓	✓		✓			✓	✓	
Cloud Support	✓	✓			✓				✓	✓
SaaS					✓	✓	✓	✓		✓
Side-by-side embeddings	✓		✓*	✓*					✓*	
Multiple embeddings	✓	✓	✓	✓	✓		✓	✓**		
Zoom in / out		✓	✓		✓		✓	✓	✓	✓
Gene coexpression	✓				✓					
Proportion plots	✓			✓					✓	
Multi-gene bubbleplot / heatmap	✓		✓	✓		✓	✓			
Differential expression		✓						✓		
Export images as PDF / PNG	✓							✓	✓	
h5ad file	✓	✓				✓	✓		✓	
loom file	✓			✓	✓	✓	✓	✓	✓	
SCE object	✓		✓							
Seurat object	✓					✓	✓			
txt / csv	✓***					✓	✓	✓	✓	✓
Platform	R	Python	R	Python	Python	Python	Python	R/Java/ Python	R	Python

S1.2 Key functionalities of different packages

We first compared the key functionalities of different visualisation tools:

- Web sharing refers to the ability to host the data visualisation on a web page. As mentioned, all the selected tools support web sharing to enable the online sharing of complex scRNA-seq datasets for non-computational researchers.
- SC analysis indicates if the tool also supports single-cell analysis such as dimension reduction and clustering etc. Only ASAP, scSVA and SPRING support single-cell analysis while the remaining tools (including ShinyCell) are dedicated tools for scRNA-seq data visualisation. We decided not to include

single-cell analysis capability as there are existing comprehensive single-cell analysis packages e.g. Seurat[10] and Scater[11].

- Docker indicates whether a docker image is provided with the tool. This makes it easier to ship and run applications across different machines and platforms. However, we did not include this functionality as ShinyCell has very few dependencies and is already relatively easy to install.
- Cloud support corresponds to availability of instructions to deploy the web applications on a public cloud. This is present in ShinyCell, which uses the R Shiny backend that can be easily deployed online.
- SaaS refers to the inclusion of a hosted version of the tool where users can upload their data to use the tool directly as a service.

S1.3 Display / UI options of different packages

We next compared the way data is being displayed and user interface (UI) options between different packages:

- Side-by-side embeddings: A distinguishing feature of ShinyCell is that it naturally allows for the side-by-side visualisation of low-dimensional embeddings coloured by two single-cell metadata / gene expressions. This is useful for comparing the metadata or gene expression of single cells on one panel while highlighting the single-cell annotation or other information on the second panel. ShinyCell is the only package to have this by default while other visualisation tools typically only show a single information on low-dimensional embeddings.
- Multiple embeddings: Most tools (including ShinyCell) allow for toggling between different low-dimensional embeddings. This is important as different dimension reduction techniques can emphasise certain underlying structure of the single-cell data. Notably, tools that support single-cell analysis generally do not allow for custom low-dimensional embeddings not calculated by the tool, which can make it difficult to integrate with other performed analysis.
- Zoom in / out: Some tools allow for users to zoom in into the visualisation to inspect certain parts of the data more closely. This is absent in ShinyCell as the inclusion of such interactivity may slow the app down. However, users can circumvent this by using ShinyCell to export PDFs, which can then be zoomed in without loss of resolution.
- Gene coexpression plots: It is useful to visualise the coexpression of two genes simultaneously on low-dimensional embeddings to ensure that the same single cell is expressing the two genes. Notably, this is not a common feature and is only present on ShinyCell and SCope.
- Proportion plots: ShinyCell allows for users to plot the composition / proportion of cells across different groups of cells. For example, this can be used to know the library composition in different clusters. Again, this is not a common feature and is only present on ShinyCell, Loom viewer and scSVA.

- Multi-gene bubbleplot / heatmap: Some tools (including ShinyCell) allow for the visualisation of the gene expression of multiple genes through the use of heatmaps or bubbleplots.
- Differential expression: A few tools support on-the-fly differential expression (DE) between groups of single cells. We have omitted this functionality in ShinyCell as such analysis can take a long time to complete especially with a large number of single-cells. More importantly, there are numerous existing DE methods and it is not realistic to implement all of them.
- Export images as PDF / PNG: Another key aspect of ShinyCell is the ability to export the visualisations as PDF or PNG files to be included in manuscripts or presentations. The resolution and image size can be adjusted to suit the purpose. This is again in line with our philosophy of making the data accessible to non-computational researchers, allowing them to present the data easily. Notably, ShinyCell is the only visualisation-dedicated tool that supports exporting images as PDF / PNG.

S1.4 Input data formats supported by different packages

Lastly, we compared the different input data formats supported by the different packages. For scRNA-seq data, there are several common data formats:

- h5ad file format is an extension of the HDF5 file format, designed to store and organise large amounts of data. h5ad extends the original format by allowing for annotations of the variables / features and observations / single cells. h5ad files are associated with the anndata python package and commonly used in python-based single-cell tools e.g. scanpy[12].
- loom file format is also based on the HDF5 file format adhering to a specific structure incorporating row and column attributes. Notably, many atlases e.g. Human Cell Atlas[13], directly provide datasets in the loom file format.
- SCE object is a R-based data object for storing data from single-cell experiments. SCE objects allow for the storage of low-dimensional embeddings, single-cell metadata, spike-in information and other metadata associated with genes and libraries. SCE objects are common input file formats for many R-based single-cell tools e.g. Scater[11] and Splatter[14].
- Seurat object is also based on the R programming language and is specifically designed for the popular R-based single-cell processing and analysis package Seurat[10].
- txt / csv files are generic plain text formats to store gene expression matrices and single-cell metadata. These files may also be compressed using gzip or other compression techniques to reduce the file size.

The loom file format is the most frequently supported format amongst the selected packages. In contrast, both R-based SCE and Seurat formats are the least supported despite the Seurat and Scater pipeline being commonplace in single-cell analysis. ShinyCell supports all the common single-cell data formats including h5ad,

loom, SCE and Seurat formats. Furthermore, we provide a tutorial with some basic single-cell analysis to convert plain-text gene expression matrices into Seurat objects for use with ShinyCell. Also, ShinyCell is the only tool that supports both SCE and Seurat objects as inputs, filling the gap in enabling interactive visualization of single-cell data processed via R-based pipelines. Furthermore, ShinyCell is written in R, providing a seamless workflow for bioinformaticians who predominantly work using the R programming language.

S1.5 Key features / optimisations in ShinyCell

Overall, ShinyCell incorporates several key features:

- Web sharing and cloud support: To reiterate, one of the main goals of ShinyCell is to enable the sharing of complex single-cell data to non-computational researchers. Thus, it is important to ensure that the ShinyCell-created interactive app can be deployed online and instructions are also provided on our github page.
- Side-by-side embeddings: Through user feedback, we found that side-by-side visualisation of low-dimensional embeddings allows for users to quickly and simultaneously compare single-cell metadata / gene expression of different groups of cells.
- Inclusion of less common visualisations: ShinyCell includes (i) coexpression plots and (ii) proportion plots, which are only found in one or two of the other compared visualisation tools. However, such visualisations are useful in (i) identifying if the same single cell expresses two genes simultaneously and (ii) knowing the library composition across different clusters respectively. These plots provide additional information on top of low-dimensional embeddings, which is commonplace in all single-cell visualisation tools.
- Export images as PDF / PNG: To enable sharing of data, ShinyCell allows users to export the visualisation in high resolution PDFs or PNGs, which can then be incorporated into presentations or manuscripts.
- Support for all common single-cell data formats: From our comparison of the supported input file formats, ShinyCell is the only tool that supports all four common single-cell data formats, namely h5ad, loom, SCE and Seurat.
- Seamless R-based pipelines: ShinyCell is also the only tool that supports both R-based SCE and Seurat objects as input except for ShinyCell. Also, ShinyCell is written in R, allowing for seamless integration into R-based single-cell analysis pipelines.

ShinyCell also incorporates several optimisations to speed up the ShinyCell-created interactive apps with great emphasis on the storage of the gene expression:

- ShinyCell stores the gene expression matrix using the HDF5 file format, which greatly reduces the memory footprint of ShinyCell-created interactive apps.

This is because the file format allows for partial loading of the gene expression data onto memory.

- We further optimised the chunking in HDF5 to allow faster access of gene expression data. The chunking specifies how the data is stored on disk and subsequently accessed. We have set the chunk size to be the gene expression of each gene i.e. a chunk size of (1,M) where M is the number of single cells. This is because we only visualise one or several genes at one time and the expression across all single cells will be required.
- Furthermore, the gene expression matrix is stored in single precision format instead of the typical double precision as the former is sufficient for visualisation purposes

References

1. David, F. P. A., Litovchenko, M., Deplancke, B. & Gardeux, V. ASAP 2020 update: an open, scalable and interactive web-based portal for (single-cell) omics analyses. *Nucleic Acids Res.* **48**, W403–W414 (2020).
2. Chan Zuckerberg Initiative. *cellxgene*. (https://github.com/chanzuckerberg/cellxgene).
3. Rue-Albrecht, K., Marini, F., Soneson, C. & Lun, A. T. L. iSEE: Interactive SummarizedExperiment Explorer. *F1000Res.* **7**, 741 (2018).
4. Linnarsson Lab. *loom-viewer*. (https://github.com/linnarsson-lab/loom-viewer).
5. Davie, K. *et al.* A Single-Cell Transcriptome Atlas of the Aging Drosophila Brain. *Cell* **174**, 982–998.e20 (2018).
6. Tabaka, M., Gould, J. & Regev, A. scSVA: an interactive tool for big data visualization and exploration in single-cell omics. *bioRxiv* (2019).
7. Feng, D., Whitehurst, C. E., Shan, D., Hill, J. D. & Yue, Y. G. Single Cell Explorer, collaboration-driven tools to leverage large-scale single cell RNA-seq data. *BMC Genomics* **20**, 676 (2019).
8. Weinreb, C., Wolock, S. & Klein, A. M. SPRING: a kinetic interface for visualizing high dimensional single-cell expression data. *Bioinformatics* **34**, 1246–1248 (2018).
9. Speir, M. L., Bhaduri, A., Markov, N. S. & Moreno, P. UCSC Cell Browser: Visualize Your Single-Cell Data. *bioRxiv* (2020).
10. Butler, A., Hoffman, P., Smibert, P., Papalexi, E. & Satija, R. Integrating single-cell transcriptomic data across different conditions, technologies, and species. *Nat. Biotechnol.* **36**, 411–420 (2018).
11. McCarthy, D. J., Campbell, K. R., Lun, A. T. L. & Wills, Q. F. Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R. *Bioinformatics* **33**, 1179–1186 (2017).
12. Wolf, F. A., Angerer, P. & Theis, F. J. SCANPY: large-scale single-cell gene expression data analysis. *Genome Biol.* **19**, 15 (2018).
13. Aviv, R. *et al.* The human cell atlas. *Elife* **6**, (2017).
14. Zappia, L., Phipson, B. & Oshlack, A. Splatter: simulation of single-cell RNA sequencing data. *Genome Biol.* **18**, 174 (2017).

S2. Tutorials for creating ShinyCell interactive applications

Here, we also included PDFs of various ShinyCell tutorials, including:

- Github readme, which includes Installation, Quick Start Guide and FAQ
- Tutorial for customising ShinyCell aesthetics
- Tutorial for creating a ShinyCell app containing several single-cell datasets
- Tutorial on other supported file formats (h5ad / loom / SCE)
- Tutorial on processing plain-text gene expression matrices
- Instructions on how to deploy ShinyCell apps online

ShinyCell package

`ShinyCell` is a R package that allows users to create interactive Shiny-based web applications to visualise single-cell data via (i) visualising cell information and/or gene expression on reduced dimensions e.g. UMAP, (ii) visualising the coexpression of two genes on reduced dimensions, (iii) visualising the distribution of continuous cell information e.g. nUMI / module scores using violin plots / box plots, (iv) visualising the composition of different clusters / groups of cells using proportion plots and (v) visualising the expression of multiple genes using bubbleplots / heatmap. Examples of ShinyCell-generated shiny apps for single and multi datasets can be found at http://shinycell1.ddnetbio.com and http://shinycell2.ddnetbio.com respectively.

Key features of `ShinyCell` include:

- Written in R and uses the Shiny package, allowing for easy sharing on online platforms e.g. shinyapps.io and Amazon Web Services (AWS) or be hosted via Shiny Server
- Supports all of the major single-cell data formats (h5ad / loom / Seurat / SingleCellExperiment) and we also include a simple tutorial to process plain-text gene expression matrices
- Web interface have low memory footprint due to the use of hdf5 file system to store the gene expression. Only the expression of genes that are plotted are loaded into memory
- Inclusion of less common single-cell visualisations, namely coexpression plots and proportion plots that provide additional information on top of low-dimensional embeddings
- Users can export visualisations as PDF or PNG images for presentation or publication use
- Ability to include multiple single-cell datasets into a single Shiny web app
- It is easy to use and customise aethetsics e.g. label names and colour palettes. In the simplest form, ShinyCell can convert an input single-cell data into a Shiny app with five lines of code (see Quick Start Guide)

This readme is broken down into the following sections:

- Installation on how to install `ShinyCell`
- Quick Start Guide to rapidly deploy a shiny app with a few lines of code
- Frequently Asked Questions

There are also additional tutorials as follows:

- Tutorial for customising ShinyCell aesthetics
- Tutorial for creating a ShinyCell app containing several single-cell datasets
- Tutorial on other supported file formats (h5ad / loom / SCE)
- Tutorial on processing plain-text gene expression matrices
- Instructions on how to deploy ShinyCell apps online

Installation

First, users can run the following code to check if the packages required by `ShinyCell` exist and install them if required:

```
reqPkg = c("data.table", "Matrix", "hdf5r", "reticulate", "ggplot2",
           "gridExtra", "glue", "readr", "RColorBrewer", "R.utils", "Seurat")
newPkg = reqPkg[!(reqPkg %in% installed.packages()[,"Package"])]
if(length(newPkg)){install.packages(newPkg)}
```

```
# If you are using h5ad file as input, run the code below as well
# reticulate::py_install("anndata")
```

Furthermore, on the system where the Shiny app will be deployed, users can run the following code to check if the packages required by the Shiny app exist and install them if required:

```
reqPkg = c("shiny", "shinyhelper", "data.table", "Matrix", "DT", "hdf5r",
           "reticulate", "ggplot2", "gridExtra", "magrittr", "ggdendro")
newPkg = reqPkg[!(reqPkg %in% installed.packages()[,"Package"])]
if(length(newPkg)){install.packages(newPkg)}
```

`ShinyCell` can then be installed from GitHub as follows:

```
devtools::install_github("SGDDNB/ShinyCell")
```

Quick Start Guide

In short, the `ShinyCell` package takes in an input single-cell object and generates a ShinyCell config `scConf` containing labelling and colour palette information for the single-cell metadata. The ShinyCell config and single-cell object are then used to generate the files and code required for the shiny app.

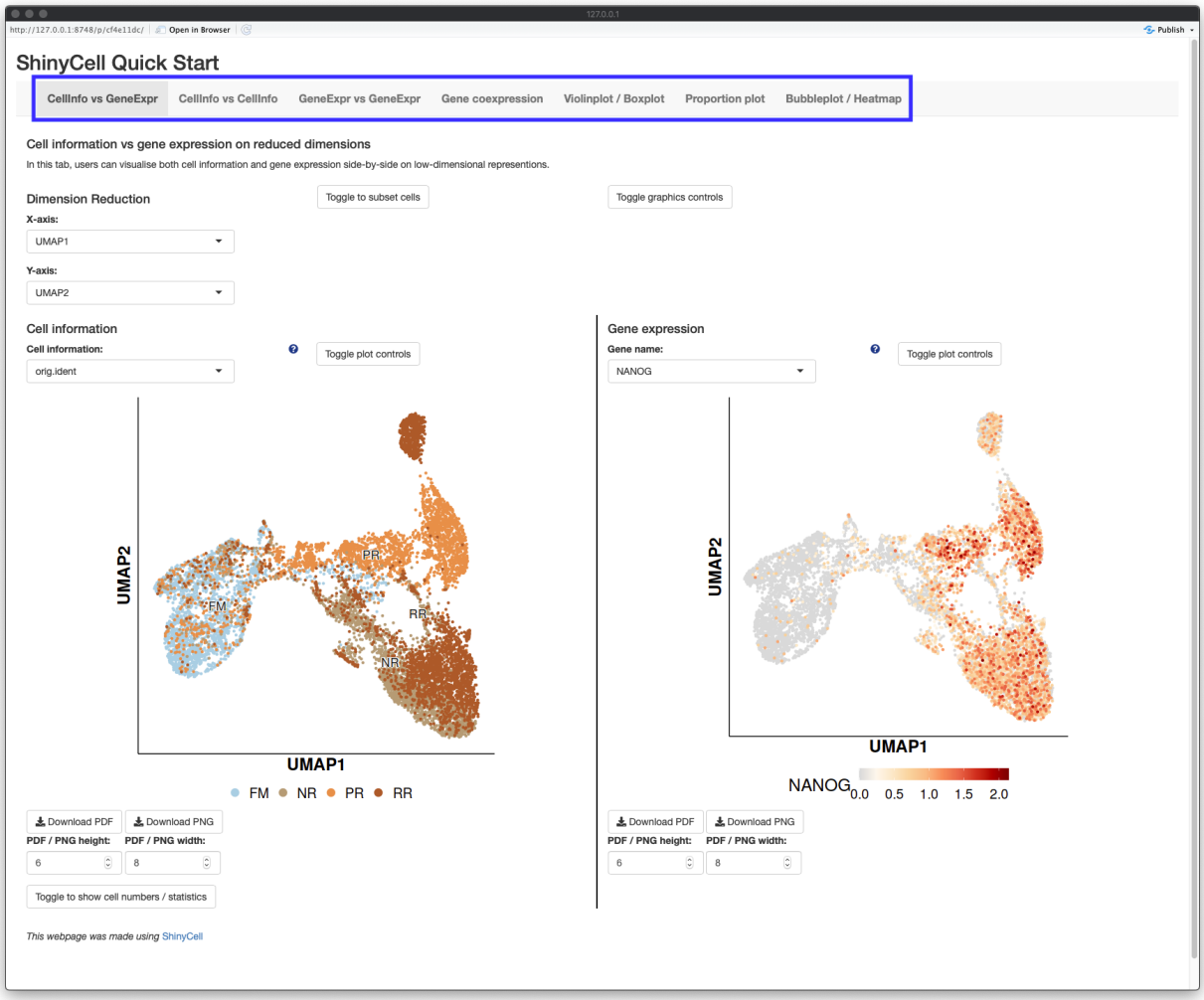
In this example, we will use single-cell data (Seurat object) containing intermediates collected during the reprogramming of human fibroblast into induced pluripotent stem cells using the RSeT media condition, taken from Liu, Ouyang, Rossello et al. Nature (2020). The Seurat object can be downloaded here.

A shiny app can then be quickly generated using the following code:

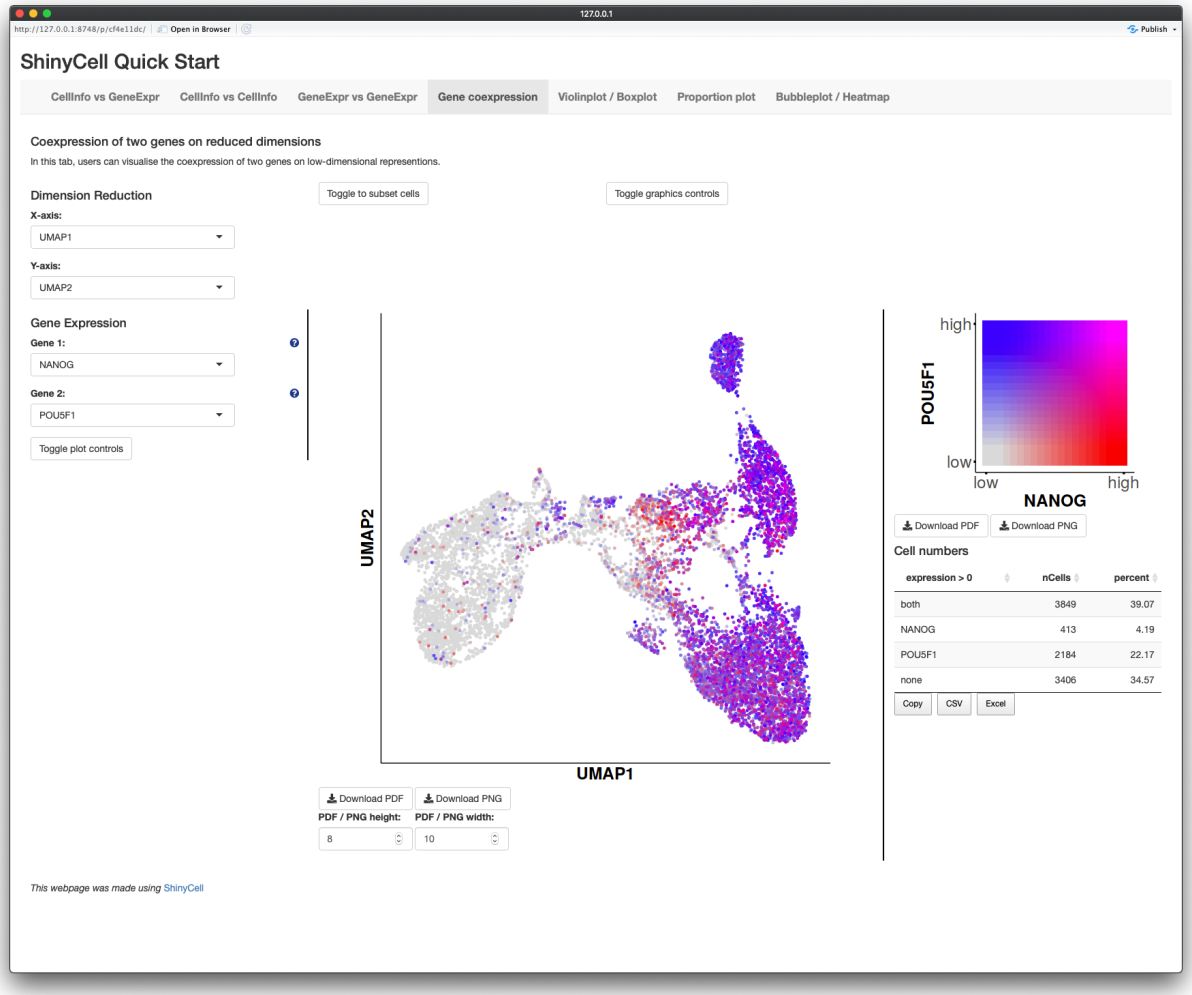
```
library(Seurat)
library(ShinyCell)

getExampleData()                        # Download example dataset (~200 MB)
seu = readRDS("readySeu_rset.rds")
scConf = createConfig(seu)
makeShinyApp(seu, scConf, gene.mapping = TRUE,
             shiny.title = "ShinyCell Quick Start")
```

The generated shiny app can then be found in the `shinyApp/` folder (which is the default output folder). To run the app locally, use RStudio to open either `server.R` or `ui.R` in the shiny app folder and click on "Run App" in the top right corner. The shiny app can also be deployed online via online platforms e.g. shinyapps.io and Amazon Web Services (AWS) or be hosted via Shiny Server. For further details, refer to Instructions on how to deploy ShinyCell apps online.

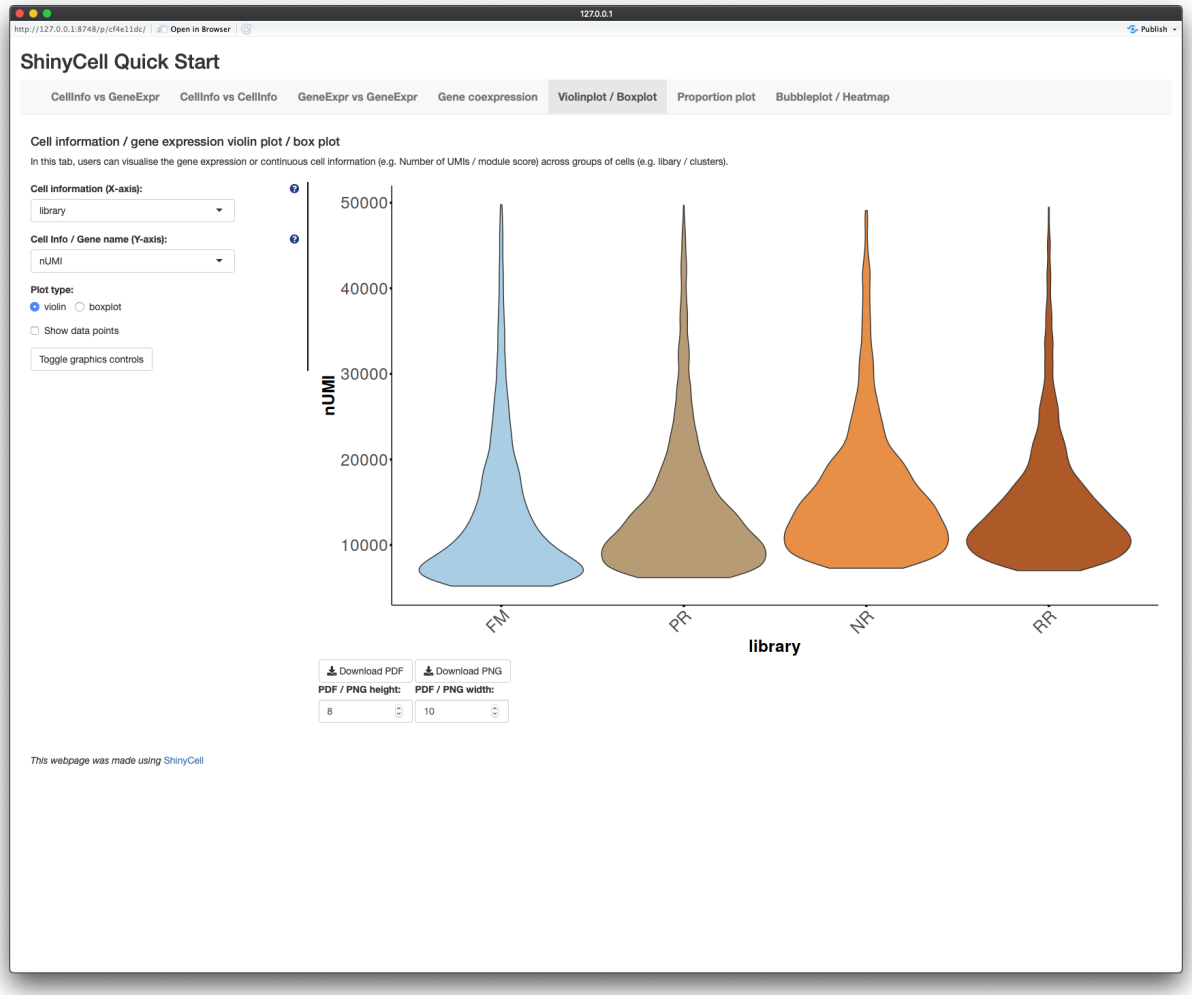
The shiny app contains seven tabs (highlighted in blue box), with the opening page showing the first tab "CellInfo vs GeneExpr" (see below), plotting both cell information and gene expression side-by-side on reduced dimensions e.g. UMAP. Users can click on the toggle on the bottom left corner to display the cell numbers in each cluster / group and the number of cells expressing a gene. The next two tabs are similar, showing either two cell information side-by-side (second tab: "CellInfo vs CellInfo") or two gene expressions side-by-side (third tab: "GeneExpr vs GeneExpr").



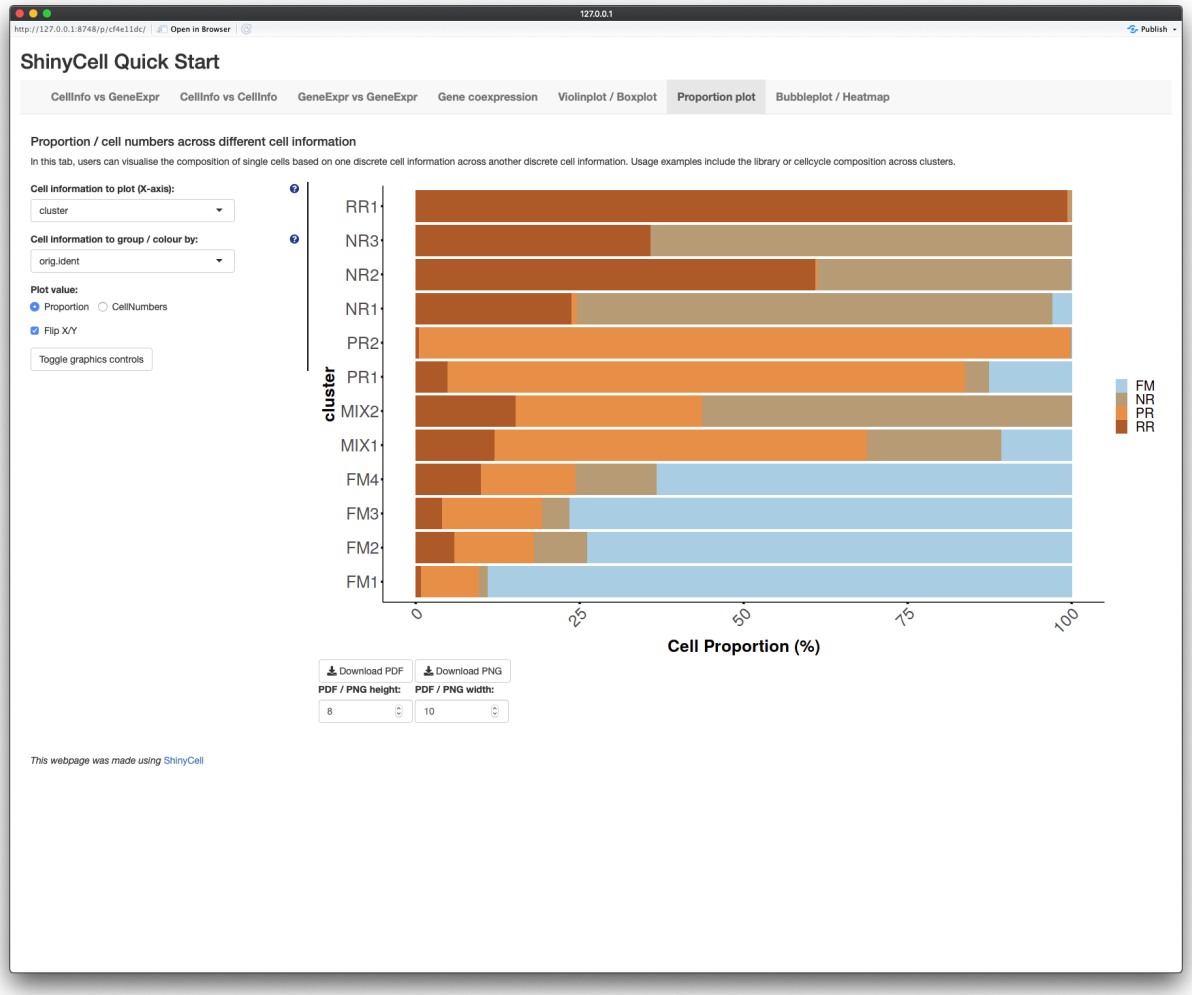
The fourth tab "Gene coexpression" blends the gene expression of two genes, given by two different colour hues, onto the same reduced dimensions plot. Furthermore, the number of cells expressing either or both genes are given.



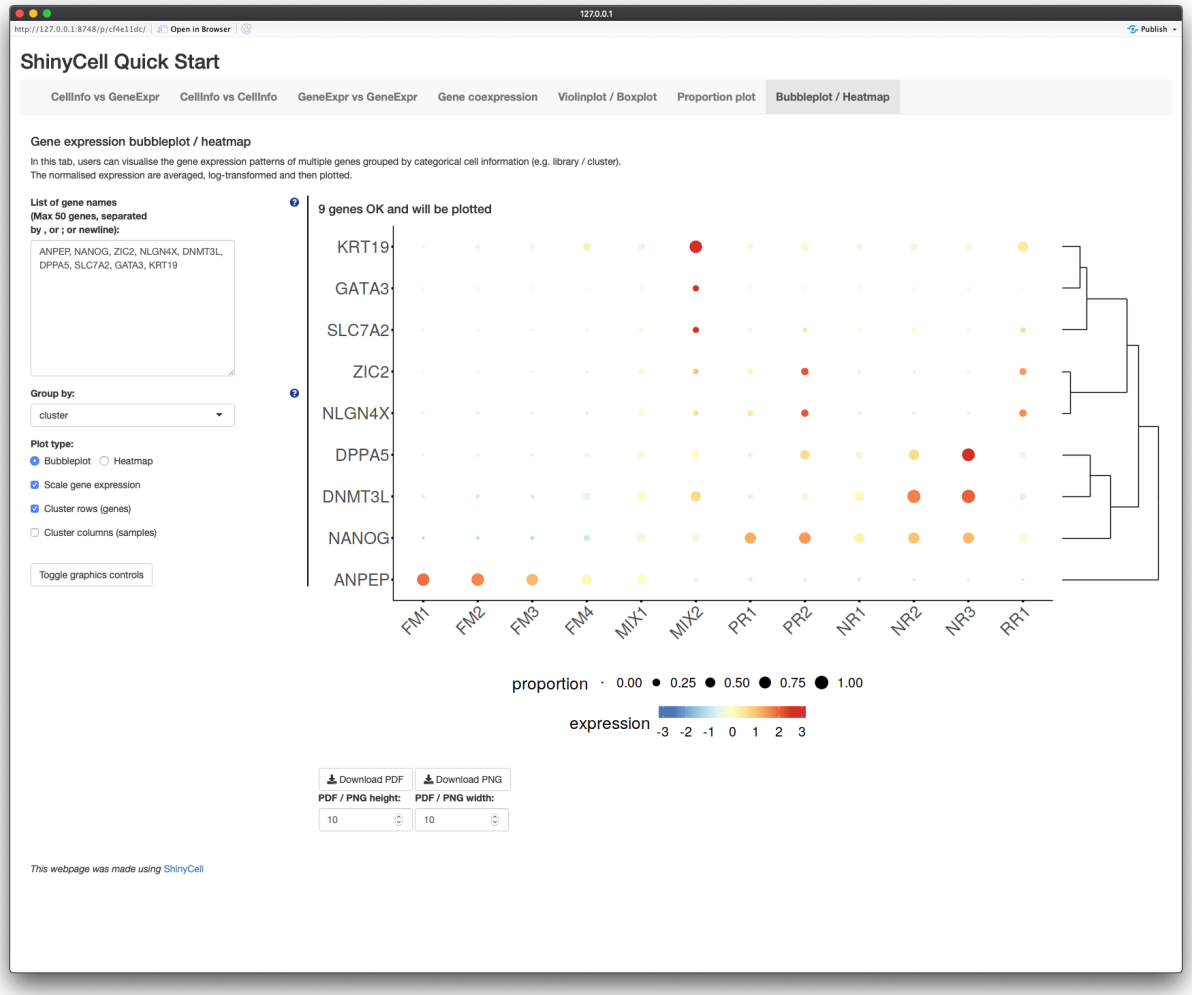
The fifth tab "Violinplot / Boxplot" plots the distribution of continuous cell information e.g. nUMI or module scores or gene expression across each cluster / group using violin plots or box plots.



The sixth tab "Proportion plot" plots the composition of different clusters / groups of cells using proportion plots. Users can also plot the cell numbers instead of proportions.



The seventh tab "Bubbleplot / Heatmap" allows users to visualise the expression of multiple genes across each cluster / group using bubbleplots / heatmap. The genes (rows) and groups (columns) can be furthered clustered using hierarchical clustering.



Frequently Asked Questions

- Q: How much memory / storage space does `ShinyCell` and the Shiny app consume?
  - A: The Shiny app itself consumes very little memory and is meant to be a heavy-duty app where multiple users can access the app simultaneously. Unlike typical R objects, the entire gene expression matrix is stored on disk and *not on memory* via the hdf5 file system. Also, the hdf5 file system offers superior file compression and takes up less storage space than native R file formats such as rds / Rdata files.
  - A: It should be noted that a large amount of memory is required when *building* the Shiny app. This is because the whole Seurat / SCE object has to be loaded onto memory and additional memory is required to generate the required files. From experience, a typical laptop with 8GB RAM can handle datasets around 30k cells while 16GB RAM machines can handle around 60k-70k cells.
- Q: I have generated additional dimension reductions (e.g. force-directed layout / MDS / monocle2 etc.) and would like to include them into the Shiny app. How do I do that?
  - A: `ShinyCell` automatically retrieves dimension reduction information from the Seurat or SCE object. Thus, the additional dimension reductions have to be added into the Seurat or SCE object before running `ShinyCell`.

  - For Seurat objects, users can refer "Storing a custom dimensional reduction calculation" in https://satijalab.org/seurat/v3.0/dim_reduction_vignette.html
  - For SCE objects, users can refer to https://bioconductor.org/packages/devel/bioc/vignettes/SingleCellExperiment/inst/doc/intro.html#3_Adding_low-dimensional_representations
- Q: I have both RNA and integrated data in my Seurat object. How do I specify which gene expression assay to plot in the Shiny app?
  - A: Only one gene expression assay can be visualised per dataset. To specify the assay, use the `gex.assay = "integrated` argument in the `makeShinyApp()` or `makeShinyFiles()` functions. If users want to visualise both gene expression, they have to treat each assay as an individual dataset and include multiple datasets into a single shiny app, following the Tutorial for creating a ShinyCell app containing several single-cell datasets

Tutorial for customising ShinyCell aesthetics and other settings

John F. Ouyang

Feb 2021

Here, we present a detailed walkthrough on how `ShinyCell` can be used to create a Shiny app from single-cell data objects. In particular, we will focus on how users can customise what metadata is to be included, their labels and colour palettes. A live version of the shiny app generated here can be found at shinycell1.ddnetbio.com.

To demonstrate, we will use single-cell data (Seurat object) containing intermediates collected during the reprogramming of human fibroblast into induced pluripotent stem cells using the RSeT media condition, taken from Liu, Ouyang, Rossello et al. Nature (2020). The Seurat object can be downloaded here.

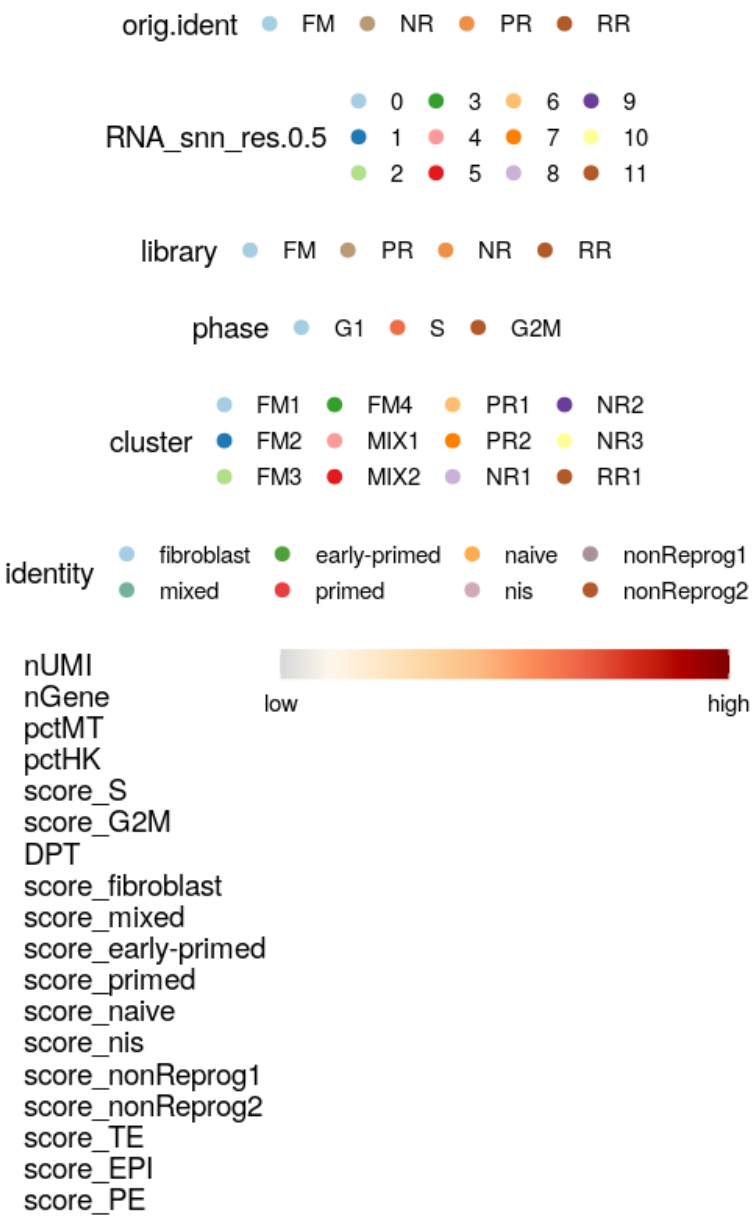
Load data and create ShinyCell configuration

First, we will load the Seurat object and run `createConfig()` to create a ShinyCell configuration `scConf`. The `scConf` is a data.table containing (i) the single-cell metadata to display on the Shiny app, (ii) ordering of factors / categories for categorical metadata e.g. library / cluster and (iii) colour palette associated with each metadata. Thus, `scConf` acts as an "instruction manual" to build the Shiny app without modifying the original single-cell data.

```
library(Seurat)
library(ShinyCell)

# Create ShinyCell config
getExampleData()                          # Download example dataset (~200 MB)
seu <- readRDS("readySeu_rset.rds")
scConf = createConfig(seu)
```

To visualise the contents of the Shiny app prior to building the actual app, we can run `showLegend()` to display the legends associated with all the single-cell metadata. This allows users to visually inspect which metadata to be shown on the Shiny app. This is useful for identifying repetitive metadata and checking how factors / categories for categorical metadata will look in the eventual Shiny app. Categorical metadata and colour palettes are shown first, followed by continuous metadata which are shown collectively.

```
showLegend(scConf)
```



Add / remove / modify metadata and colour palette

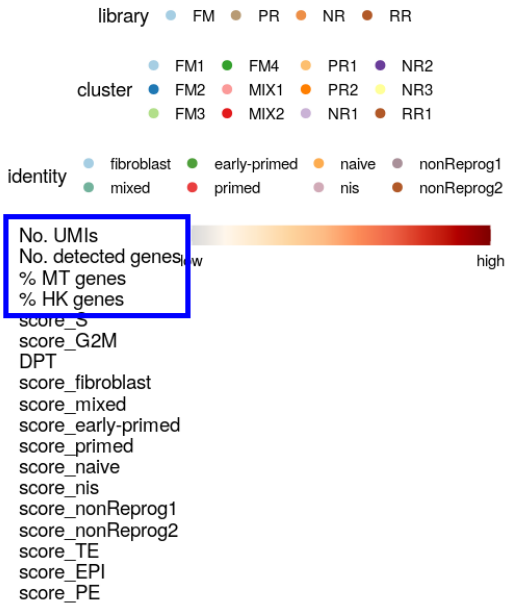
It is possible to modify `scConf` directly but this might be prone to error. Thus, we provided numerous convenience functions to modify `scConf` and ultimately the Shiny app. In this example, we note that the `orig.ident` and `library` as well as `RNA_snn_res.0.5` and `cluster` metadata are similar. To exclude metadata from the Shiny app, we can run `delMeta()`. Furthermore, we can modify how the names of metadata appear by running `modMetaName()`. In this case, we changed the names of some metadata to make them more meaningful.

By default, colours for categorical metadata are generated by interpolating colours from the "Paired" colour palette in the RColorBrewer package. To modify the colour palette, we can

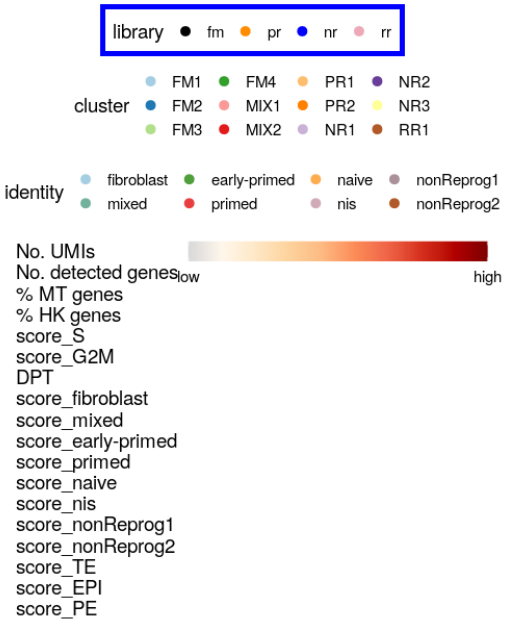
run `modColours()`. Here, we changed the colours for the library metadata to match that in the publication. It is also possible to modify the labels for each category via `modLabels()`. For example, we changed the labels for the library metadata from upper case to lower case. After modifying `scConf`, it is reccomended to run `showLegend()` to inspect the changes made.

```
# Delete excessive metadata and rename some metadata
scConf = delMeta(scConf, c("orig.ident", "RNA_snn_res.0.5", "phase"))
scConf = modMetaName(scConf,
                     meta.to.mod = c("nUMI", "nGene", "pctMT", "pctHK"),
                     new.name = c("No. UMIs", "No. detected genes",
                                  "% MT genes", "% HK genes"))
showLegend(scConf)

# Modify colours and labels
scConf = modColours(scConf, meta.to.mod = "library",
                    new.colours= c("black", "darkorange", "blue", "pink2"))
scConf = modLabels(scConf, meta.to.mod = "library",
                   new.labels = c("fm", "pr", "nr", "rr"))
showLegend(scConf)
```

After running
`delMeta()`
`modMetaName()`



After running
`modColours()`
`modLabels()`



Change order of appearance of metadata and defaults

Apart from `showLegend()`, users can also run `showOrder()` to display the order in which metadata will appear in the dropdown menu when selecting which metadata to plot in the Shiny app. A table will be printed showing the actual name of the metadata in the single-cell object and the display name in the Shiny app. The metadata type (either categorical or continuous) is also provided with the number of categories "nlevels". Finally, the "default" column indicates which metadata are the primary and secondary default.

```
showOrder(scConf)
```

	actual name	display name	type	nlevels	default
1	library	library	cat.	4	1
2	nUMI	No. UMIs	cont.	0	
3	nGene	No. detected genes	cont.	0	
4	pctMT	% MT genes	cont.	0	
5	pctHK	% HK genes	cont.	0	
6	score_S	score_S	cont.	0	
7	score_G2M	score_G2M	cont.	0	
8	cluster	cluster	cat.	12	2
9	identity	identity	cat.	8	
10	DPT	DPT	cont.	0	
11	score_fibroblast	score_fibroblast	cont.	0	
12	score_mixed	score_mixed	cont.	0	
13	score_early-primed	score_early-primed	cont.	0	
14	score_primed	score_primed	cont.	0	
15	score_naive	score_naive	cont.	0	
16	score_nis	score_nis	cont.	0	
17	score_nonReprog1	score_nonReprog1	cont.	0	
18	score_nonReprog2	score_nonReprog2	cont.	0	
19	score_TE	score_TE	cont.	0	
20	score_EPI	score_EPI	cont.	0	
21	score_PE	score_PE	cont.	0	

Here, we introduce a few more functions that might be useful in modifying the Shiny app. Users can add metadata back via `addMeta()`. The newly added metadata (in this case, the phase metadata) is appended to the bottom of the list as shown by `showOrder()`. Next, we

can reorder the order in which metadata appear in the dropdown menu in the Shiny app via `reorderMeta()`. Here, we shifted the phase metadata up the list. Finally, users can change the default metadata to plot via `modDefault()`. Again, it is reccomended to run `showOrder()` frequently to check how the metadata is changed.

```
# Add metadata back, reorder, default
scConf = addMeta(scConf, "phase", seu)
showOrder(scConf)
scConf = reorderMeta(scConf, scConf$ID[c(1:5,22,6:21)])
showOrder(scConf)
scConf = modDefault(scConf, "library", "identity")
showOrder(scConf)
```

`addMeta()`

	actual name	display name	type	nlevels	default
1	library	library	cat.	4	1
2	nUMI	No. UMIs	cont.	0	
3	nGene	No. detected genes	cont.	0	
4	pctMT	% MT genes	cont.	0	
5	pctHK	% HK genes	cont.	0	
6	score_S	score_S	cont.	0	
7	score_G2M	score_G2M	cont.	0	
8	cluster	cluster	cat.	12	2
9	identity	identity	cat.	8	
10	DPT	DPT	cont.	0	
11	score_fibroblast	score_fibroblast	cont.	0	
12	score_mixed	score_mixed	cont.	0	
13	score_early-primed	score_early-primed	cont.	0	
14	score_primed	score_primed	cont.	0	
15	score_naive	score_naive	cont.	0	
16	score_nIs	score_nIs	cont.	0	
17	score_nonReprog1	score_nonReprog1	cont.	0	
18	score_nonReprog2	score_nonReprog2	cont.	0	
19	score_TE	score_TE	cont.	0	
20	score_EPI	score_EPI	cont.	0	
21	score_PE	score_PE	cont.	0	
22	phase	phase	cat.	3	

`reorderMeta()`

	actual name	display name	type	nlevels	default
1	library	library	cat.	4	1
2	nUMI	No. UMIs	cont.	0	
3	nGene	No. detected genes	cont.	0	
4	pctMT	% MT genes	cont.	0	
5	pctHK	% HK genes	cont.	0	
6	phase	phase	cat.	3	
7	score_S	score_S	cont.	0	
8	score_G2M	score_G2M	cont.	0	
9	cluster	cluster	cat.	12	2
10	identity	identity	cat.	8	
11	DPT	DPT	cont.	0	
12	score_fibroblast	score_fibroblast	cont.	0	
13	score_mixed	score_mixed	cont.	0	
14	score_early-primed	score_early-primed	cont.	0	
15	score_primed	score_primed	cont.	0	
16	score_naive	score_naive	cont.	0	
17	score_nIs	score_nIs	cont.	0	
18	score_nonReprog1	score_nonReprog1	cont.	0	
19	score_nonReprog2	score_nonReprog2	cont.	0	
20	score_TE	score_TE	cont.	0	
21	score_EPI	score_EPI	cont.	0	
22	score_PE	score_PE	cont.	0	

`modDefault()`

	actual name	display name	type	nlevels	default
1	library	library	cat.	4	1
2	nUMI	No. UMIs	cont.	0	
3	nGene	No. detected genes	cont.	0	
4	pctMT	% MT genes	cont.	0	
5	pctHK	% HK genes	cont.	0	
6	phase	phase	cat.	3	
7	score_S	score_S	cont.	0	
8	score_G2M	score_G2M	cont.	0	
9	cluster	cluster	cat.	12	
10	identity	identity	cat.	8	2
11	DPT	DPT	cont.	0	
12	score_fibroblast	score_fibroblast	cont.	0	
13	score_mixed	score_mixed	cont.	0	
14	score_early-primed	score_early-primed	cont.	0	
15	score_primed	score_primed	cont.	0	
16	score_naive	score_naive	cont.	0	
17	score_nIs	score_nIs	cont.	0	
18	score_nonReprog1	score_nonReprog1	cont.	0	
19	score_nonReprog2	score_nonReprog2	cont.	0	
20	score_TE	score_TE	cont.	0	
21	score_EPI	score_EPI	cont.	0	
22	score_PE	score_PE	cont.	0	

Generate Shiny app

After modifying `scConf` to one's satisfaction, we are almost ready to build the Shiny app. Prior to building the Shiny app, users can run `checkConfig()` to check if the `scConf` is in the right format. This is especially useful if users have manually modified the `scConf`. Users can also add a footnote to the Shiny app, which can be plain text or the citation for the dataset. To input a citation, a list is required, populating various information e.g. authors, title, year. An example of including the citation as the Shiny app footnote is provided below.

```
# Build shiny app
checkConfig(scConf, seu)
citation = list(
  author  = "Liu X., Ouyang J.F., Rossello F.J. et al.",
  title   = "",
  journal = "Nature",
  volume  = "586",
  page    = "101-107",
  year    = "2020",
```

```
  doi     = "10.1038/s41586-020-2734-6",
  link    = "https://www.nature.com/articles/s41586-020-2734-6")
```

Now, we can build the shiny app! A few more things need to be specified here. In this example, the Seurat object uses Ensembl IDs and we would like to convert them to more user-friendly gene symbols in the Shiny app. `ShinyCell` can do this conversion (for human and mouse datasets) conveniently by specifying `gene.mapping = TRUE`. If your dataset is already in gene symbols, you can leave out this argument to not perform the conversion. Furthermore, `ShinyCell` uses the "RNA" assay and "data" slot in Seurat objects as the gene expression data. If you have performed any data integration and would like to use the integrated data instead, please specify `gex.assay = "integrated"`. Also, default genes to plot can be specified where `default.gene1` and `default.gene2` corresponds to the default genes when plotting gene expression on reduced dimensions while `default.multigene` contains the default set of multiple genes when plotting bubbleplots or heatmaps. If unspecified, `ShinyCell` will automatically select some genes present in the dataset as default genes.

```
makeShinyApp(seu, scConf, gene.mapping = TRUE,
             gex.assay = "RNA", gex.slot = "data",
             shiny.title = "ShinyCell Tutorial",
             shiny.dir = "shinyApp/", shiny.footnotes = citation,
             default.gene1 = "NANOG", default.gene2 = "DNMT3L",
             default.multigene = c("ANPEP","NANOG","ZIC2","NLGN4X","DNMT3L",
                                   "DPPA5","SLC7A2","GATA3","KRT19"))
```

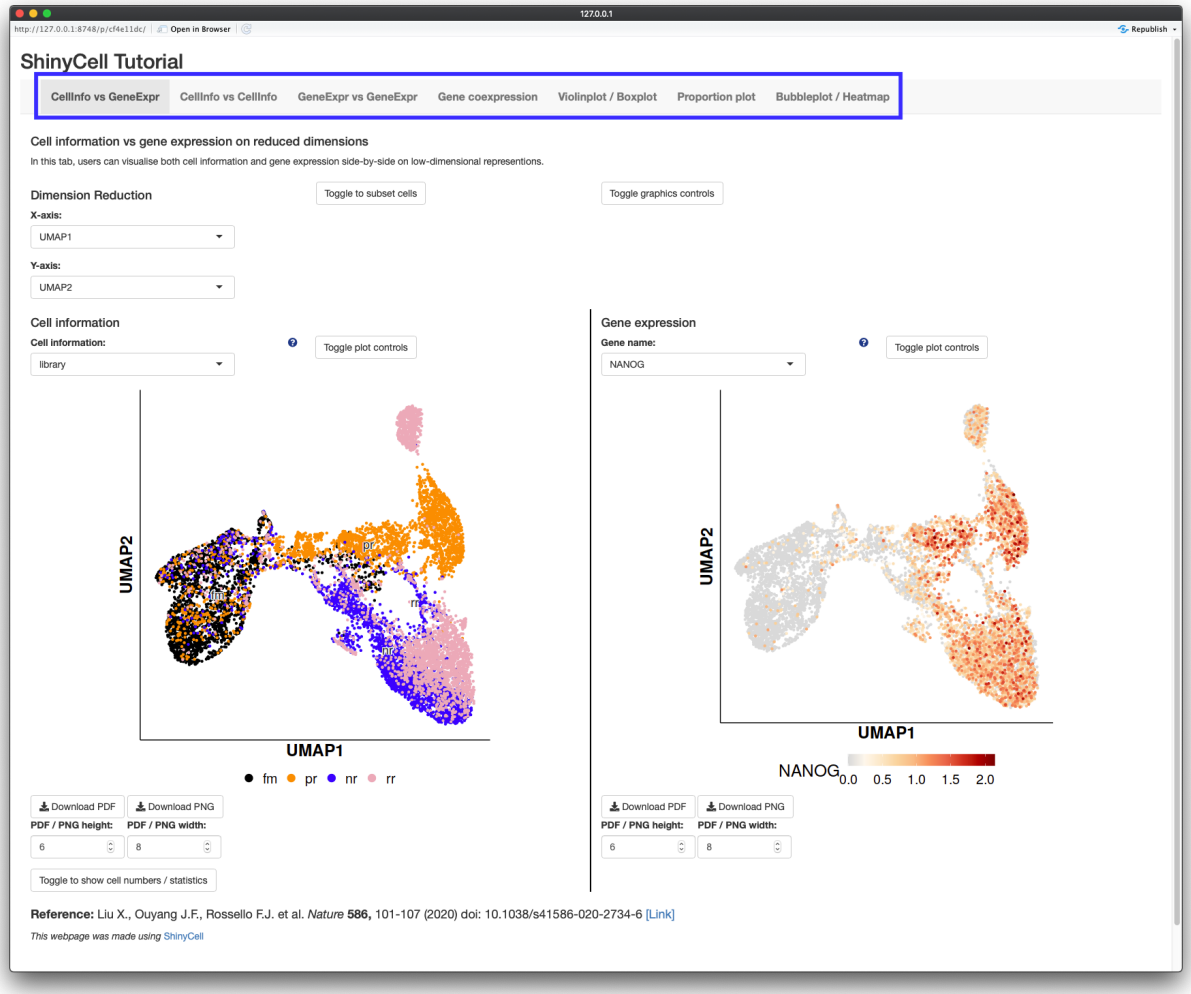
Under the hood, `makeShinyApp()` does two things: generate (i) the data files required for the Shiny app and (ii) the code files, namely `server.R` and `ui.R`. The generated files can be found in the `shinyApp/` folder. To run the app locally, use RStudio to open either `server.R` or `ui.R` in the shiny app folder and click on "Run App" in the top right corner. The shiny app can also be deployed online via online platforms e.g. shinyapps.io and Amazon Web Services (AWS) or be hosted via Shiny Server. For further details, refer to Instructions on how to deploy ShinyCell apps online.

Different visualisations in the Shiny app

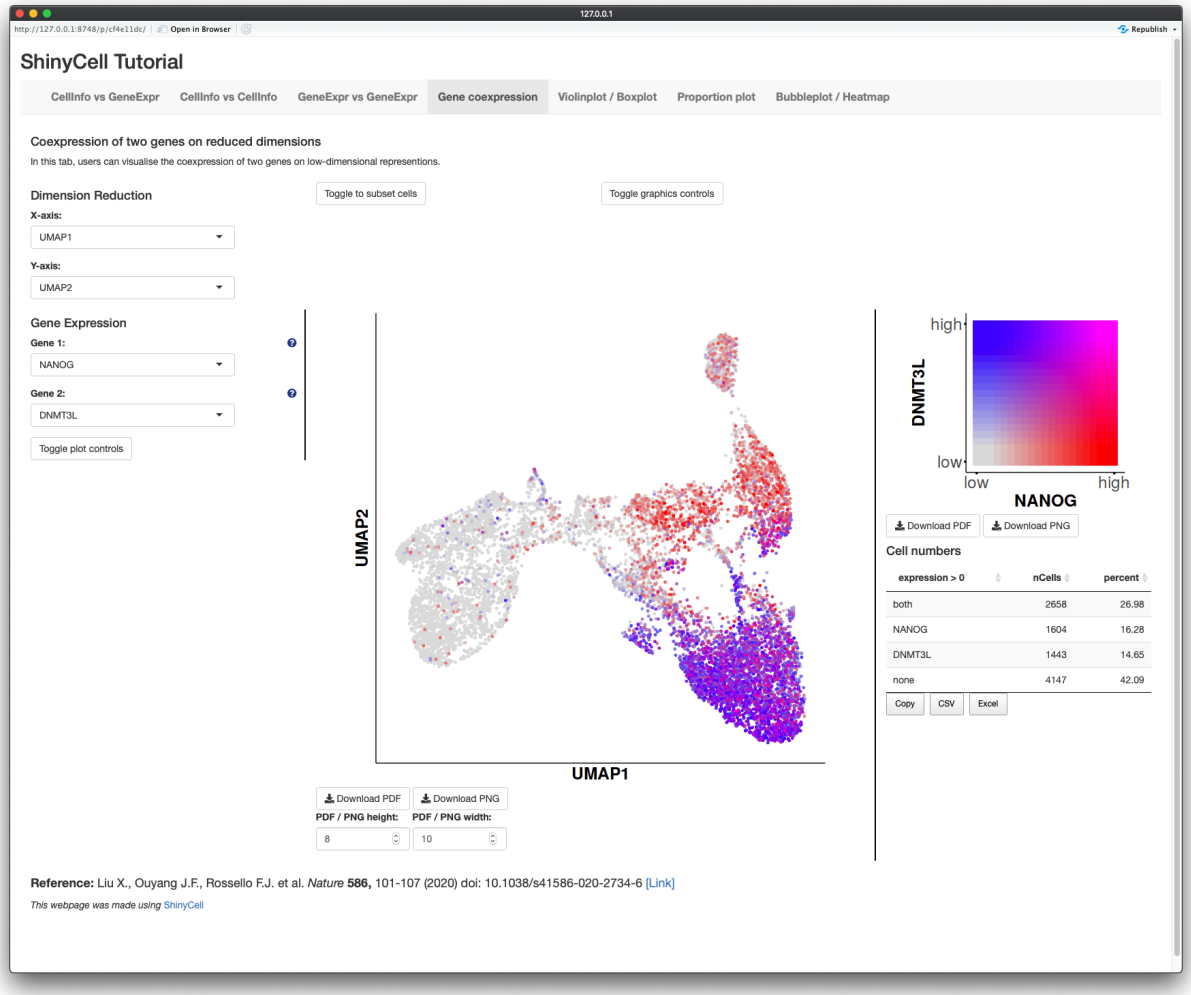
With the Shiny app, users can interactively explore their single-cell data, varying the cell information / gene expression to plot. Furthermore, these plots can be exported into PDF / PNG for presentations / publications. Users can also click on the "Toggle graphics controls" or "Toggle plot controls" to fine-tune certain aspects of the plots e.g. point size. A live version of this shiny app can be found at shinycell1.ddnetbio.com.

The shiny app contains seven tabs (highlighted in blue box), with the opening page showing the first tab "CellInfo vs GeneExpr" (see below), plotting both cell information and gene expression side-by-side on reduced dimensions e.g. UMAP. Users can click on the toggle on the bottom left corner to display the cell numbers in each cluster / group and the number of cells expressing a gene. The next two tabs are similar, showing either two cell information

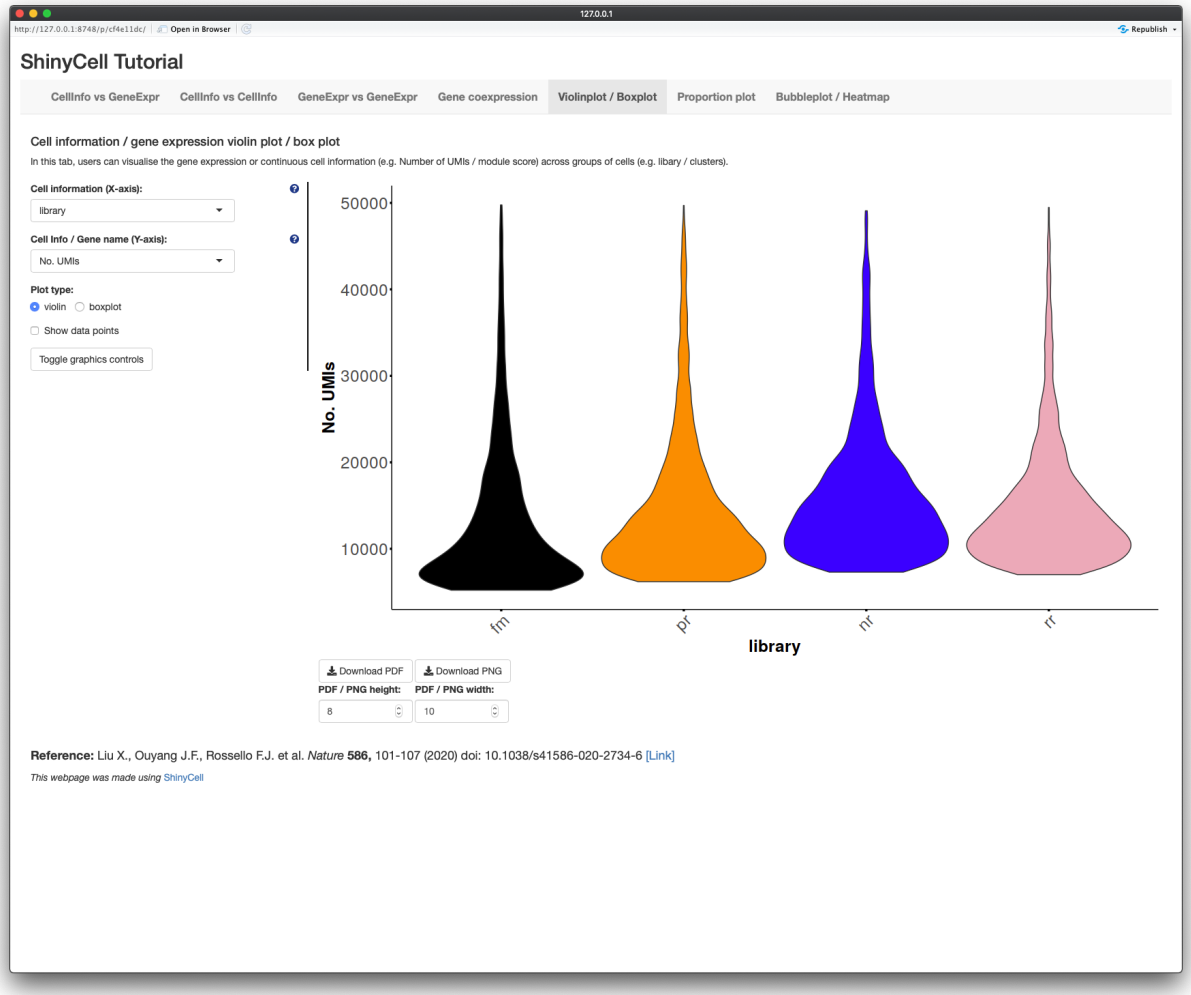
side-by-side (second tab: “CellInfo vs CellInfo”) or two gene expressions side-by-side (third tab: “GeneExpr vs GeneExpr”).



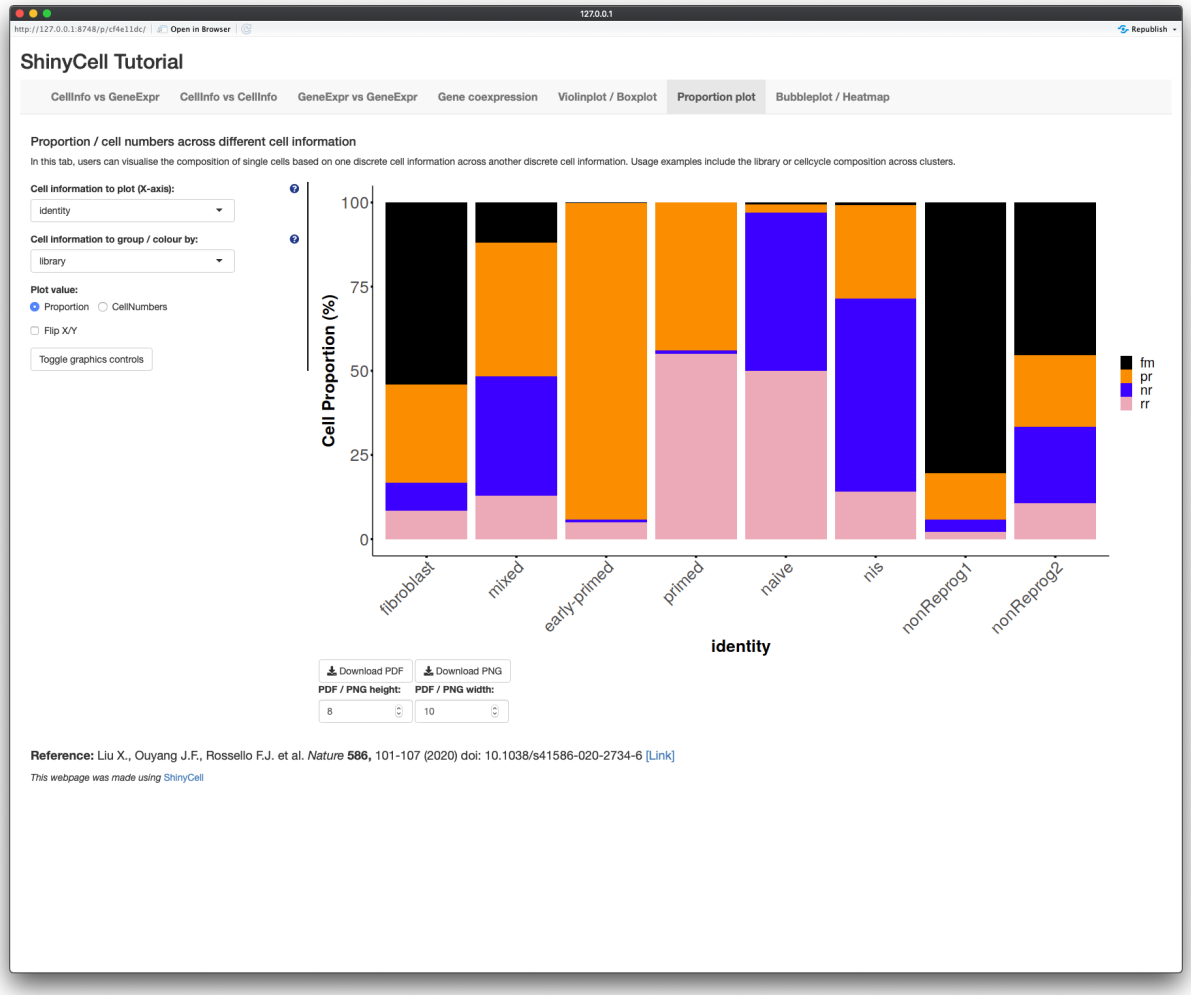
The fourth tab “Gene coexpression” blends the gene expression of two genes, given by two different colour hues, onto the same reduced dimensions plot. Furthermore, the number of cells expressing either or both genes are given.



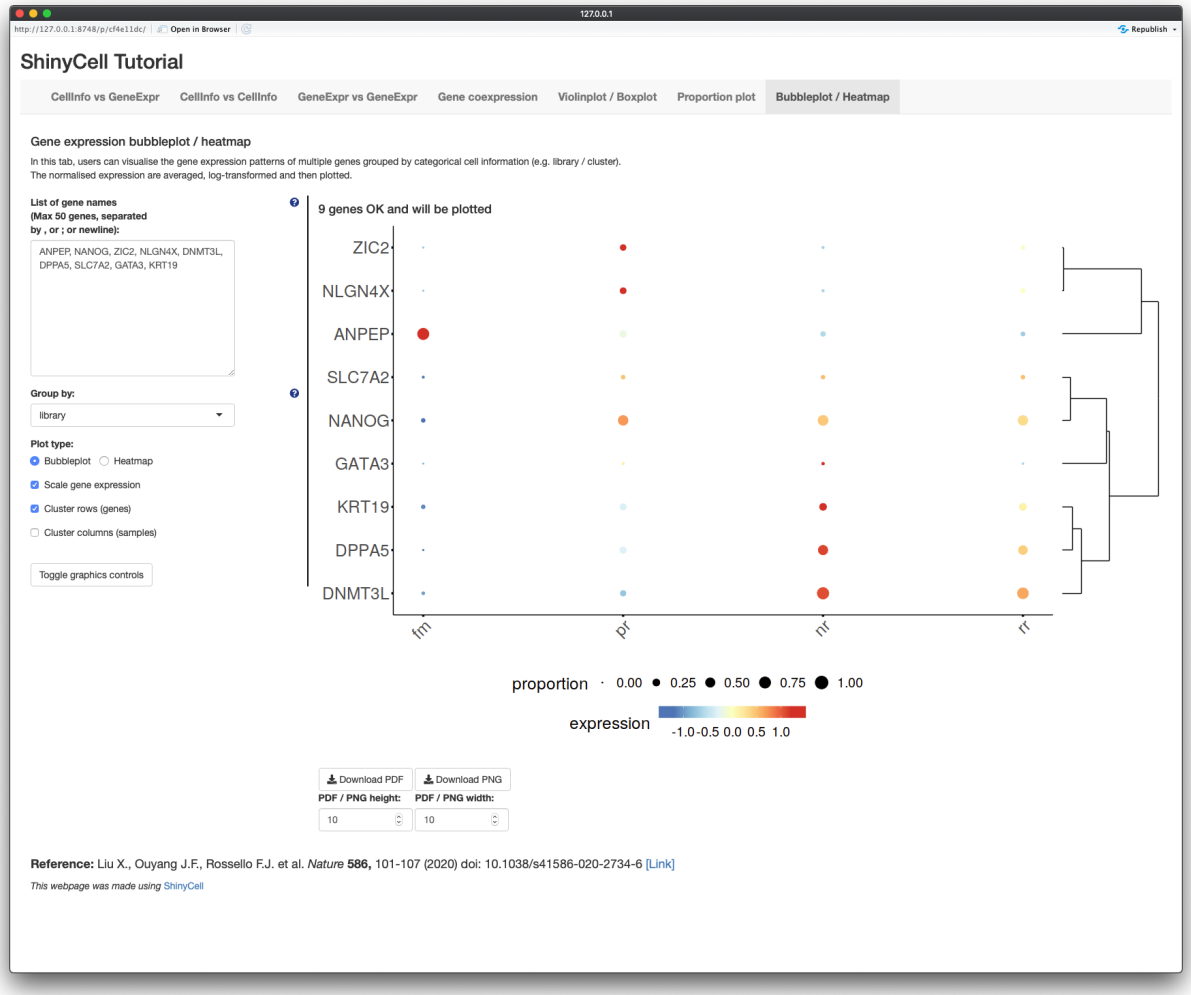
The fifth tab "Violinplot / Boxplot" plots the distribution of continuous cell information e.g. nUMI or module scores or gene expression across each cluster / group using violin plots or box plots.



The sixth tab "Proportion plot" plots the composition of different clusters / groups of cells using proportion plots. Users can also plot the cell numbers instead of proportions.



The seventh tab "Bubbleplot / Heatmap" allows users to visualise the expression of multiple genes across each cluster / group using bubbleplots / heatmap. The genes (rows) and groups (columns) can be furthered clustered using hierarchical clustering.


127.0.0.1
http://127.0.0.1:8748/p/cf4e11dc/
Open in Browser
Republish
ShinyCell Tutorial
CellInfo vs GeneExpr
CellInfo vs CellInfo
GeneExpr vs GeneExpr
Gene coexpression
Violinplot / Boxplot
Proportion plot
Bubbleplot / Heatmap
Gene expression bubbleplot / heatmap
In this tab, users can visualise the gene expression patterns of multiple genes grouped by categorical cell information (e.g. library / cluster).
The normalised expression are averaged, log-transformed and then plotted.
List of gene names
(Max 50 genes, separated
by , or ; or newline):
ANPEP, NANOG, ZIC2, NLGN4X, DNMT3L,
DPPA5, SLC7A2, GATA3, KRT19
Group by:
library
Plot type:
Bubbleplot
Heatmap
Scale gene expression
Cluster rows (genes)
Cluster columns (samples)
Toggle graphics controls
9 genes OK and will be plotted
ZIC2
NLGN4X
ANPEP
SLC7A2
NANOG
GATA3
KRT19
DPPA5
DNMT3L
fm
pr
nr
rr
proportion
0.00
0.25
0.50
0.75
1.00
expression
-1.0 -0.5 0.0 0.5 1.0
Download PDF
Download PNG
PDF / PNG height:
PDF / PNG width:
10
10
Reference: Liu X., Ouyang J.F., Rossello F.J. et al. Nature 586, 101-107 (2020) doi: 10.1038/s41586-020-2734-6 [Link]
This webpage was made using ShinyCell

Tutorial for creating a ShinyCell app containing several single-cell datasets

John F. Ouyang

Feb 2021

Users might want to include multiple single-cell datasets into a single Shiny app and `ShinyCell` provides this functionality. We will demonstrate how to create a shiny app containing two single-cell datasets. A live version of the shiny app generated here can be found at shinycell2.ddnetbio.com.

To further change the aesthetics and ordering of metadata, please refer to the Tutorial for changing ShinyCell aesthetics and other settings.

Load data

For the first example dataset, we will use scRNA-seq data (Seurat object) containing intermediates collected during reprogramming of human fibroblast into induced pluripotent stem cells using the RSeT media condition, which can be downloaded here. For the second example dataset, we will use scRNA-seq of day 21 reprogramming intermediates from the same publication, which can be downloaded here. After downloading the data, we will begin by loading the required libraries.

```
library(Seurat)
library(ShinyCell)
getExampleData("multi")      # Download multiple example datasets (~400 MB)
```

Create ShinyCell configuration and configure settings for dataset 1

To create a multi-dataset Shiny app, we need to configure the settings for each dataset separately. We will do so for the first dataset as follows. A ShinyCell configuration `scConf1` is created, followed by modifying various aspects of the Shiny app e.g. removing excessive metadata, modifying the display names of metadata and modifying the colour palettes. For a more detailed explanation on how to customise the shiny app, refer to Tutorial for customising ShinyCell aesthetics and other settings. We then run `makeShinyFiles()` to generate the files related to the first dataset. Notice that we specified `shiny.prefix = "sc1"` and this prefix is used to identify that the files containing single-cell data related to the first dataset. The remaining arguments are the same as explained in the Tutorial for customising ShinyCell aesthetics and other settings.

```
seu <- readRDS("readySeu_rset.rds")
scConf1 = createConfig(seu)
scConf1 = delMeta(scConf1, c("orig.ident", "RNA_snn_res.0.5"))
scConf1 = modMetaName(scConf1, meta.to.mod = c("nUMI", "nGene", "pctMT", "pctHK"),
                      new.name = c("No. UMIs", "No. detected genes",
                                   "% MT genes", "% HK genes"))
scConf1 = modColours(scConf1, meta.to.mod = "library",
                     new.colours= c("black", "darkorange", "blue", "pink2"))
makeShinyFiles(seu, scConf1, gex.assay = "RNA", gex.slot = "data",
               gene.mapping = TRUE, shiny.prefix = "sc1",
               shiny.dir = "shinyAppMulti/",
               default.gene1 = "NANOG", default.gene2 = "DNMT3L",
               default.multigene = c("ANPEP","NANOG","ZIC2","NLGN4X","DNMT3L",
                                     "DPPA5","SLC7A2","GATA3","KRT19"),
               default.dimred = c("UMAP_1", "UMAP_2"))
```

Create ShinyCell configuration and configure settings for dataset 2

We then repeat the same procedure for the second dataset to generate the files required for the Shiny app. Notice that we used a different prefix here `shiny.prefix = "sc2"`.

```
seu <- readRDS("readySeu_d21i.rds")
scConf2 = createConfig(seu)
scConf2 = delMeta(scConf2, c("orig.ident", "RNA_snn_res.0.5"))
scConf2 = modMetaName(scConf2, meta.to.mod = c("nUMI", "nGene", "pctMT", "pctHK"),
                      new.name = c("No. UMIs", "No. detected genes",
                                   "% MT genes", "% HK genes"))
scConf2 = modColours(scConf2, meta.to.mod = "library",
                     new.colours= c("black", "blue", "purple"))
makeShinyFiles(seu, scConf2, gex.assay = "RNA", gex.slot = "data",
               gene.mapping = TRUE, shiny.prefix = "sc2",
               shiny.dir = "shinyAppMulti/",
               default.gene1 = "GATA3", default.gene2 = "DNMT3L",
               default.multigene = c("ANPEP","NANOG","ZIC2","NLGN4X","DNMT3L",
                                     "DPPA5","SLC7A2","GATA3","KRT19"),
               default.dimred = c("UMAP_1", "UMAP_2"))
```

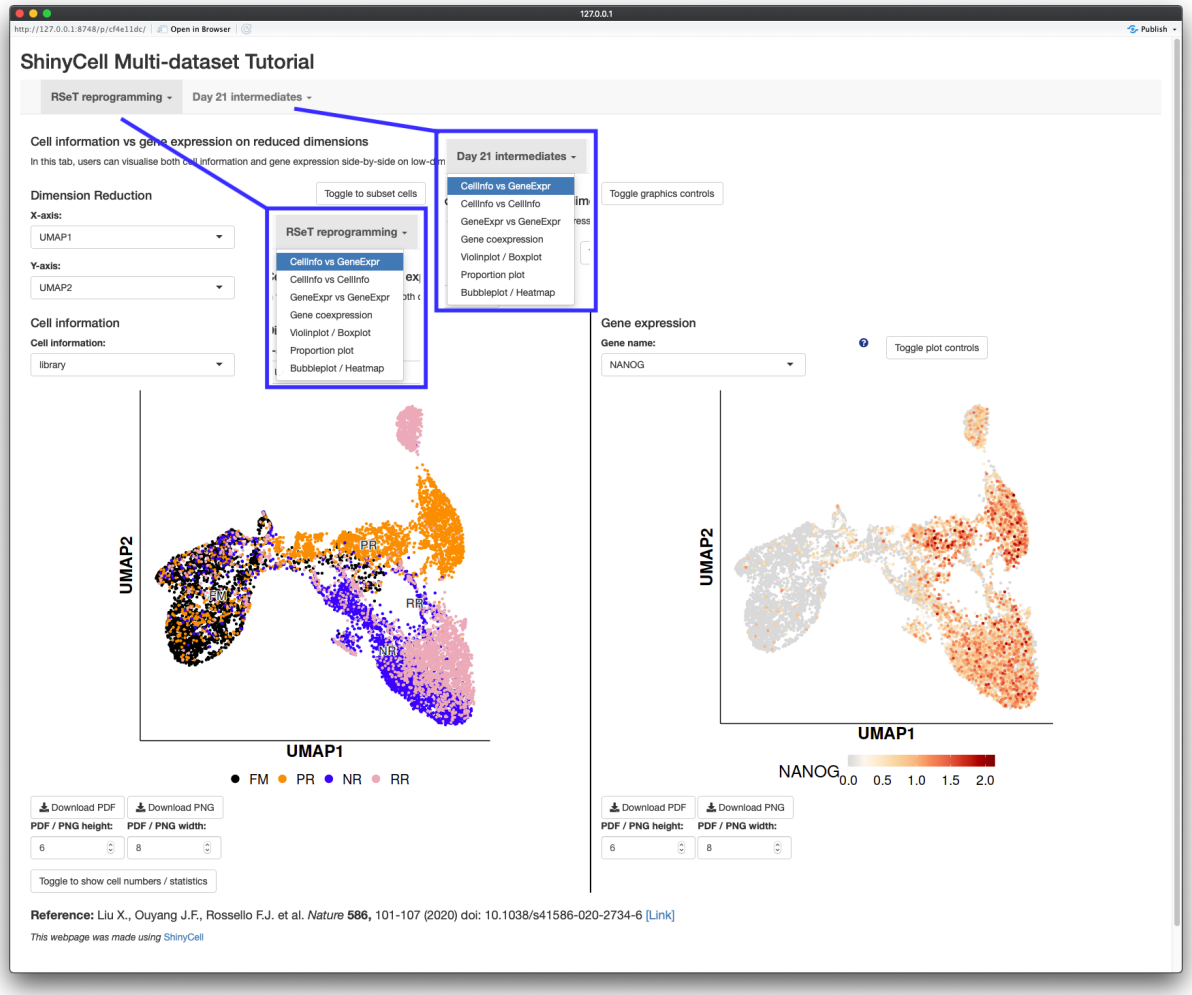
Generate code for Shiny app

We can the proceed to the final part where we generate the code for the Shiny app using the `makeShinyCodesMulti()` function. To specify that two datasets will be included in this Shiny app, we input the prefixes of the two datasets `shiny.prefix = c("sc1", "sc2")`.

Also, users need to specify section headers for each dataset via the `shiny.headers` argument. The remaining arguments are the same as explained in the Tutorial for changing ShinyCell aesthetics and other settings.

```
citation = list(
  author  = "Liu X., Ouyang J.F., Rossello F.J. et al.",
  title   = "",
  journal = "Nature",
  volume  = "586",
  page    = "101-107",
  year    = "2020",
  doi     = "10.1038/s41586-020-2734-6",
  link    = "https://www.nature.com/articles/s41586-020-2734-6")
makeShinyCodesMulti(
  shiny.title = "Multi-dataset Tutorial", shiny.footnotes = citation,
  shiny.prefix = c("sc1", "sc2"),
  shiny.headers = c("RSeT reprogramming", "Day 21 intermediates"),
  shiny.dir = "shinyAppMulti/")
```

Now, we have both the data and code for the Shiny app and we can run the Shiny app. Each dataset can be found in their corresponding tabs and clicking on the tab creates a dropdown to change the type of plot to display on the Shiny app. This tutorial can be easily expanded to include three or more datasets. Users simply have to create the corresponding data files for each dataset and finally generate the code for the Shiny app. A live version of the shiny app can be found at shinycell2.ddnetbio.com.


127.0.0.1
ShinyCell Multi-dataset Tutorial
RSeT reprogramming
Day 21 intermediates
Cell information vs gene expression on reduced dimensions
Dimension Reduction
X-axis:
UMAP1
Y-axis:
UMAP2
Cell information
Cell information:
library
Toggle to subset cells
Toggle graphics controls
CellInfo vs GeneExpr
CellInfo vs CellInfo
GeneExpr vs GeneExpr
Gene coexpression
Violinplot / Boxplot
Proportion plot
Bubbleplot / Heatmap
Gene expression
Gene name:
NANOG
Toggle plot controls
UMAP2
UMAP1
FM
PR
RR
NR
FM PR NR RR
NANOG 0.0 0.5 1.0 1.5 2.0
Download PDF
Download PNG
PDF / PNG height:
PDF / PNG width:
6
8
Toggle to show cell numbers / statistics
Reference: Liu X., Ouyang J.F., Rossello F.J. et al. Nature 586, 101-107 (2020) doi: 10.1038/s41586-020-2734-6 [Link]
This webpage was made using ShinyCell

Tutorial for other supported file formats (h5ad / loom / SCE)

John F. Ouyang

Feb 2021

ShinyCell supports all four common single-cell data formats, namely h5ad, loom, SCE and Seurat. In our other tutorials, we focused on using Seurat objects as inputs. Here, we will explain how to use the other supported file formats to create shiny apps from single-cell data objects.

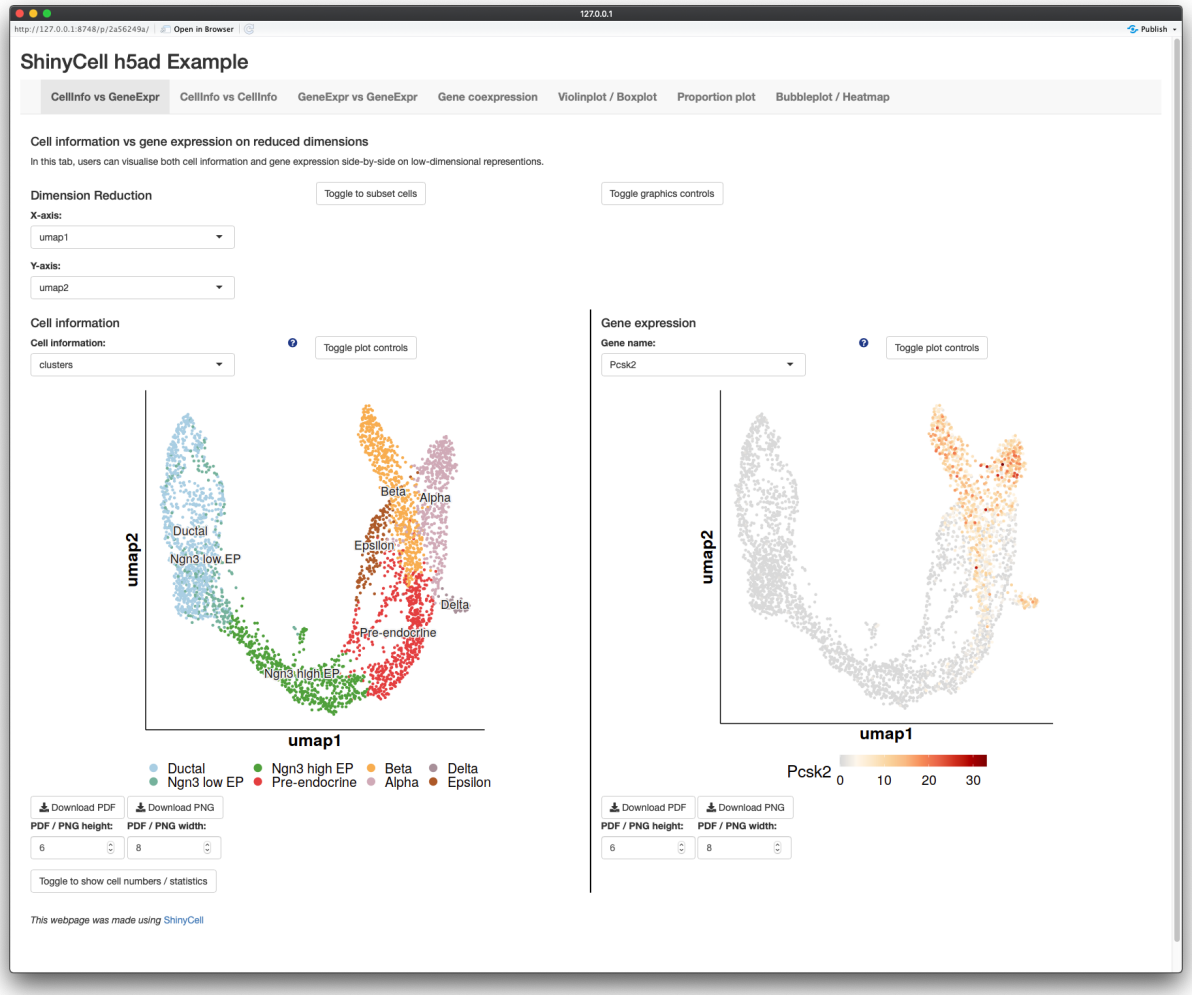
Overall, the process is identical to the other tutorials except that the file path is required for h5ad / loom files and the SCE object itself is supplied for SingleCellExperiment objects. To further change the aesthetics and ordering of metadata, please refer to the Tutorial for changing ShinyCell aesthetics and other settings. To include multiple datasets into one shiny app, please refer to the Tutorial for creating a ShinyCell app containing several single-cell datasets.

h5ad file as input

For using h5ad file as input, we will demonstrate using the scRNA-seq of endocrine development in the pancreas, taken from Bastidas-Ponce et al. Development (2018). The dataset is taken from the scVelo package tutorial.

```
library(ShinyCell)
getExampleData("h5ad")

inpFile = "endocrinogenesis_day15.h5ad"
scConf = createConfig(inpFile)
makeShinyApp(inpFile, scConf,
             default.gene1 = "Pcsk2", default.gene2 = "Dcdc2a",
             default.multigene = c("Pcsk2", "Dcdc2a", "Ank", "Gng12", "Top2a",
                                   "Pak3", "Tmem163", "Nfib", "Pim2", "Smoc1"),
             shiny.dir = "shinyAppH5ad/",
             shiny.title = "ShinyCell h5ad Example")
```

Running the above code generates a shiny app in the `shinyAppH5ad/` folder, looking like this:



loom file as input

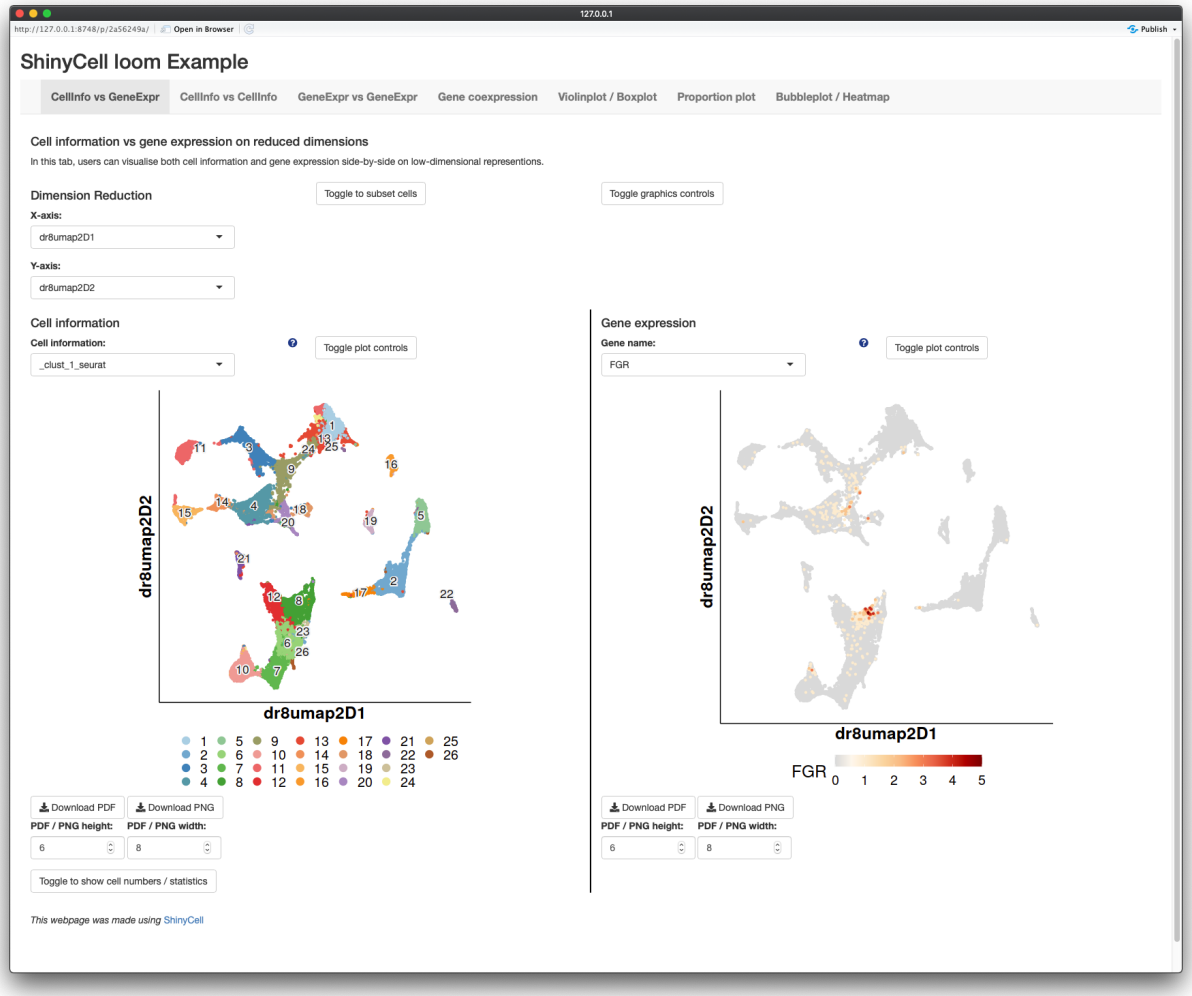
The loom file format is a hdf5-based file format that is commonly used in cell atlases e.g. Human Cell Atlas. We would like to emphasize again that ShinyCell is a visualisation tool and *not an analysis tool* with the aim of providing sharable visualisation of *already-analysed* single-cell datasets. We assume that common single-cell analysis such as dimension reduction and clustering has been performed. Thus, loom files downloaded directly from cell atlas data portals may not have these analysis performed and ShinyCell will not work accordingly.

To demonstrate using a loom file as input, we will be using the scRNA-seq of hematopoietic differentiation, taken from Setty et al. Nature Biotechnology (2019). Dimension reduction and clustering analysis of this dataset has been performed using the ASAP pipeline; key: xr7ne3.

```
library(ShinyCell)
getExampleData("loom")

inpFile <- "xr7ne3_dim_reduction_13225_output.loom"
scConf = createConfig(inpFile, meta.to.include =
                        c("_Depth","_Detected_Genes","_Mitochondrial_Content",
                          "_Protein_Coding_Content","_Ribosomal_Content",
                          "_clust_1_seurat","bundle_version","donor_organism.sex"))

# Make "_clust_1_seurat" categorical and default
scConf = makeMetaCategorical(scConf, "_clust_1_seurat", inpFile)
scConf[ID == "_clust_1_seurat"]$fRow = 4
scConf = modDefault(scConf, "_clust_1_seurat", "bundle_version")

makeShinyApp(inpFile, scConf,
             shiny.dir = "shinyAppLoom/",
             shiny.title = "ShinyCell loom Example")
```

Note that loom files do not necessarily store the categories / levels information for categorical single-cell metadata. ShinyCell tries to circumvent this by automatically factoring all metadata containing text. However, there are cases where the single-cell metadata are integers and it is difficult to automatically detect if these metadata are meant to be treated in a discrete or continuous manner. In this loom file, the seurat clusters `_clust_1_seurat` is one such example. To make the seurat clusters a categorical metadata, we ran the function `makeMetaCategorical()`. We further modified the number of rows in the legends of the seurat clusters `_clust_1_seurat` metadata to 4 and also set the metadata to be plotted by default.

Running the above code generates a shiny app in the `shinyAppLoom/` folder, looking like this:



SCE object as input

For using SCE object as input, we will demonstrate using the Seurat object used in the other tutorials. The Seurat object can be easily converted to a SCE object and the SCE object is processed by ShinyCell in the same way as its Seurat counterpart. Here, we reuse the code from the quick start guide.

```
library(Seurat)
library(ShinyCell)

getExampleData()                          # Download example dataset (~200 MB)
seu = readRDS("readySeu_rset.rds")
sce = as.SingleCellExperiment(seu)
scConf = createConfig(sce)
makeShinyApp(sce, scConf, gene.mapping = TRUE,
             shiny.title = "ShinyCell Quick Start")
```

Tutorial on processing plain-text gene expression matrices

John F. Ouyang

Feb 2021

Introduction

Apart from major single-cell data formats (h5ad / loom / Seurat / SCE), users may start with a plain-text gene expression matrix or output from the 10X cellranger pipeline. Here, we provide a simple tutorial on performing some basic single-cell analysis using the Seurat pipeline and creating a ShinyCell app using the generated Seurat object.

WARNING: We would like to emphasize that ShinyCell is a visualisation tool and *not an automated analysis tool*. So while we provide some code to perform basic single-cell analysis, it is up to the user to ensure that the Seurat analysis makes sense and agree with their own biological understanding.

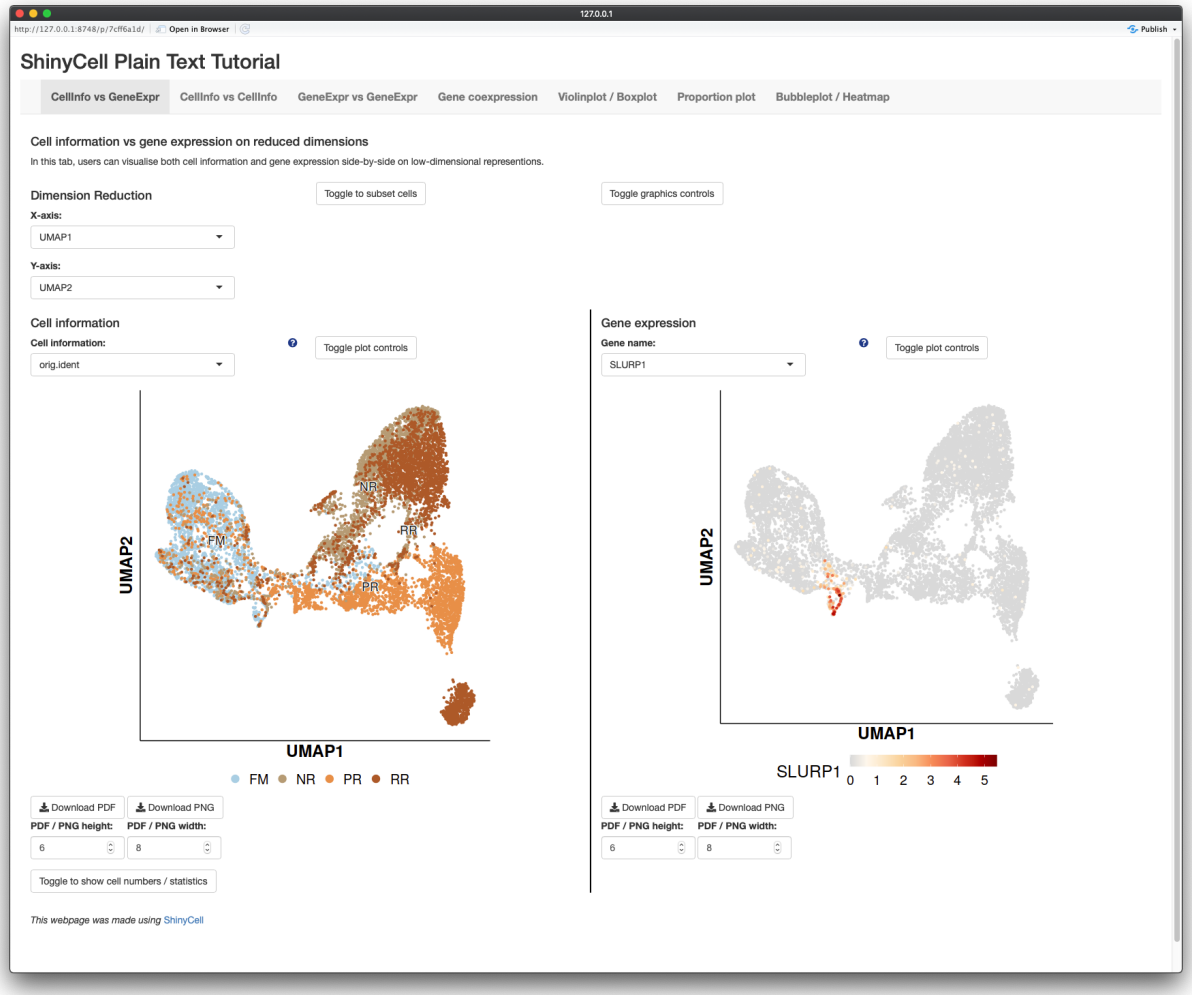
Code

The code comprises several parts. First, after loading the required packages, we will read in the plain-text gene expression matrix and convert it into a sparse matrix format. For cellranger outputs, one can run the `Read10X()` instead to read in the gene expression from the cellranger output directory. Second, we will perform some basic single-cell analysis using the Seurat pipeline. The analysis include preprocessing the data, followed by PCA and UMAP dimension reduction and then unsupervised clustering. Third, we will create a ShinyCell app using the generated Seurat object. The code is essentially the same as that in the quick start guide.

```
library(data.table)
library(Matrix)
library(Seurat)
library(ShinyCell)


## 1. Read in gene expression
getExampleData("plaintext")  # Download plain text example dataset
inpGEX = fread("test/rset.txt.gz")
inpGEX = as.matrix(inpGEX[, -1], rownames = inpGEX$V1)
inpGEX = as(inpGEX, "dgCMatrix")
# inpGEX = Read10X(data.dir = cellranger/output/directory/)
```

```
## 2. Seurat pipeline
seu = CreateSeuratObject(counts = inpGEX)
seu = NormalizeData(seu)
seu = FindVariableFeatures(seu, nfeatures = 1500)
seu = ScaleData(seu)

seu = RunPCA(seu, npcs = 30)
seu = RunUMAP(seu, dims = 1:15)
seu = FindNeighbors(seu, dims = 1:15)
seu = FindClusters(seu, resolution = 0.5, random.seed = 42)

## 3. ShinyCell
scConf = createConfig(seu)
makeShinyApp(seu, scConf, gene.mapping = TRUE,
             shiny.dir = "shinyAppPlain",
             shiny.title = "ShinyCell Plain Text Tutorial")
```

Running the above code generates a shiny app in the `shinyAppPlain/` folder, looking like this:


127.0.0.1
http://127.0.0.1:8748/p/7cff6a1d/
Open in Browser
Publish
ShinyCell Plain Text Tutorial
CellInfo vs GeneExpr
CellInfo vs CellInfo
GeneExpr vs GeneExpr
Gene coexpression
Violinplot / Boxplot
Proportion plot
Bubbleplot / Heatmap
Cell information vs gene expression on reduced dimensions
In this tab, users can visualise both cell information and gene expression side-by-side on low-dimensional representions.
Dimension Reduction
Toggle to subset cells
Toggle graphics controls
X-axis:
UMAP1
Y-axis:
UMAP2
Cell information
Cell information:
orig.ident
Toggle plot controls
NR
RR
FM
PR
UMAP2
UMAP1
FM NR PR RR
Download PDF
Download PNG
PDF / PNG height:
PDF / PNG width:
6
8
Toggle to show cell numbers / statistics
This webpage was made using ShinyCell
Gene expression
Gene name:
SLURP1
Toggle plot controls
UMAP2
UMAP1
SLURP1
0 1 2 3 4 5
Download PDF
Download PNG
PDF / PNG height:
PDF / PNG width:
6
8

Instructions on how to deploy ShinyCell apps online

John F. Ouyang

Feb 2021

The shiny app generated by ShinyCell can be easily deployed in a variety of ways, namely via (i) https://www.shinyapps.io/, (ii) an in-house server using R Shiny Server or (iii) other cloud computing platforms e.g. Amazon Web Services (AWS). The ease and availability of several cloud options is due to the use of R Shiny backend, which is designed to be easily deployed online.

shinyapps.io

One of the simplest way to deploy the shiny app is via the dedicated online service https://www.shinyapps.io/. First, one has to register for an account, which will prompt the user to create an account name `ACCOUNT` that will constitute part of the URL of the eventual shiny app. After registration, one has to setup the connection between your R / RStudio and shinyapps.io via the code below. One can then then upload the shiny app by specifying the shiny app directory `shinyApp/` and the app will be available online at `https://[ACCOUNT].shinyapps.io/shinyApp/`. For more details, refer to the shinyapps user guide.

```
install.packages('rsconnect')     # package to interface shiny apps
library(rsconnect)

rsconnect::setAccountInfo(name="<ACCOUNT>",
                          token="<TOKEN>",
                          secret="<SECRET>")
rsconnect::deployApp("shinyApp/")
```

R Shiny Server

If you have access to a linux-based server within your institution, you can deploy the shiny app on the server via R Shiny Server. This will require the installation of R and R Shiny Server on the server and the installation details of R Shiny Server can be found here.

After installing R Shiny Server, the shiny app can be deployed by placing the entire folder `shinyApp` into the directory `/srv/shiny-server/`. The shiny app can then accessed via

`[IP-address-of-server]:3838/shinyApp`. For more details, refer to the Administrator's Guide.

Depending on how the internet network is set up within your institution, it is often that the IP address of the server is only accessible within the institution's local network for security reasons. In that scenario, the shiny app is only accessible to other members in your institution and an internal shiny app can be setup for sharing with wetlab coworkers. If users wish to make the shiny app available for public access, they may have to contact their institution's IT department to lift any network restrictions.

Amazon Web Services (AWS)

Another way to deploy the shiny app online is to use cloud computing platforms such as Amazon Web Services (AWS). This will ensure that the shiny apps can be accessed publically as these services are independent from the institution's local network. Here, we will briefly outline how to deploy shiny apps on AWS. There are three main steps, namely (i) creating a new Elastic Compute Cloud (EC2) instance, (ii) installing R and R Shiny Server and (iii) deploying the shiny app.

There are several online tutorials that describe these three steps in detail, such as https://towardsdatascience.com/how-to-deploy-your-shiny-app-to-aws-856587cd412c and https://www.charlesbordet.com/en/guide-shiny-aws.