Introduction to Single-Cell Data Mining

Working notes for the single cell gene expression analytics workshop at University of Pavia

These notes include scOrange workflows and visualizations we will construct during the course.

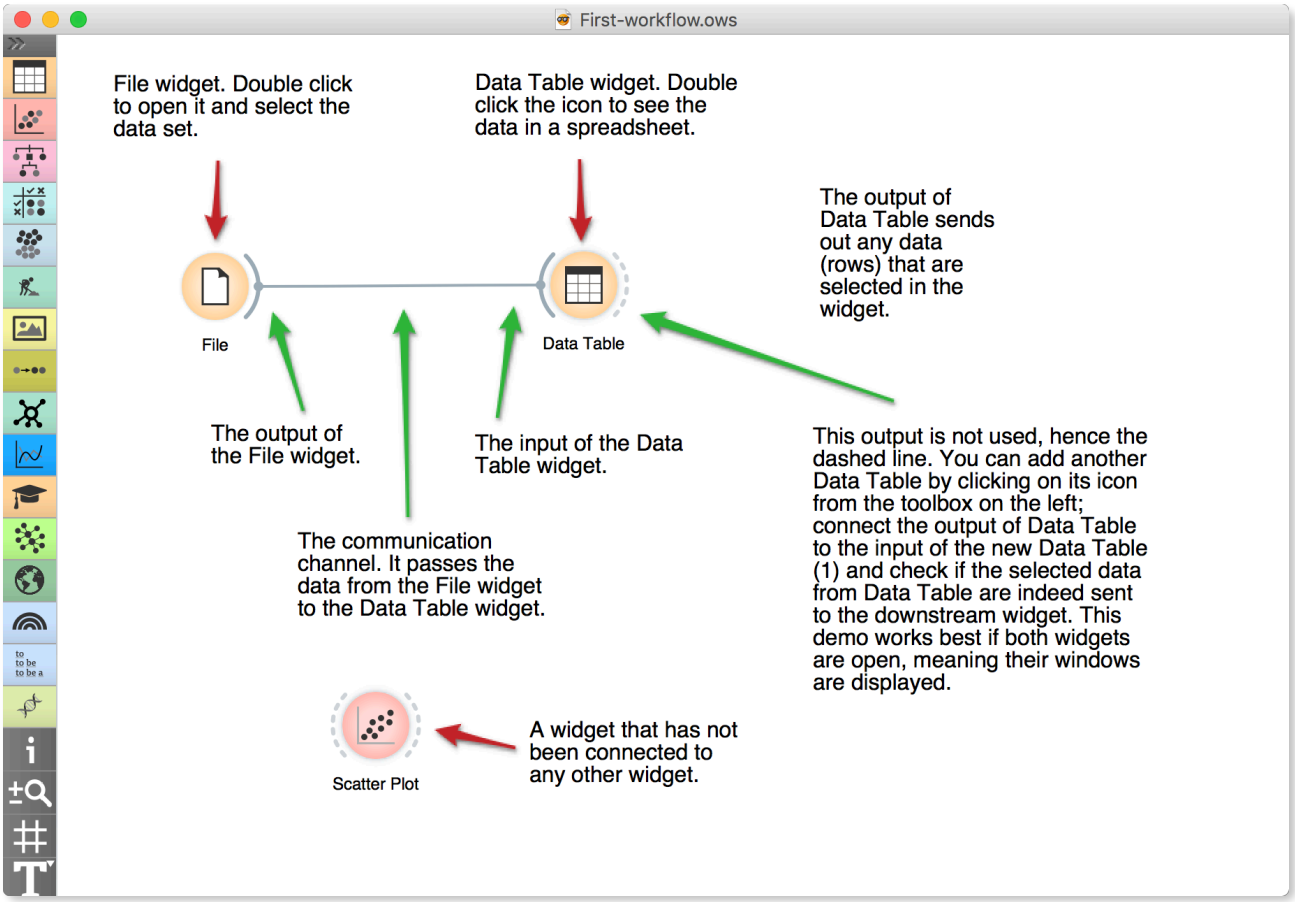
The notes were written by Martin Stražar and Blaž Zupan with a huge help from the members of the Bioinformatics Lab in Ljubljana that develop and maintain scOrange. In part, we have reused lecture notes for Orange workshops as published by the same group.

Welcome! We have designed this course for biologists interested in interactive single-cell gene expression data analysis. You will see how you can accomplish single-cell data mining tasks without programming. We will use Orange and its bioinformatics and single-cell add-ons to construct reproducible, shareable and visual data mining workflows. The same functionality is also available in application called scOrange.

If you haven't already installed Orange or scOrange, please download the installation package from https://singlecell.biolab.si. The program scOrange is a derivative of Orange, an open-source data mining suite, with a pre-installed add-on for single-cell analytics. In this notes, we will refer to both of the programs as Orange, and refer to a program we will use during the workshop as scOrange when emphasizing that the functionality we describe is not included in the Orange distribution.

Lesson 1: Workflows in Orange

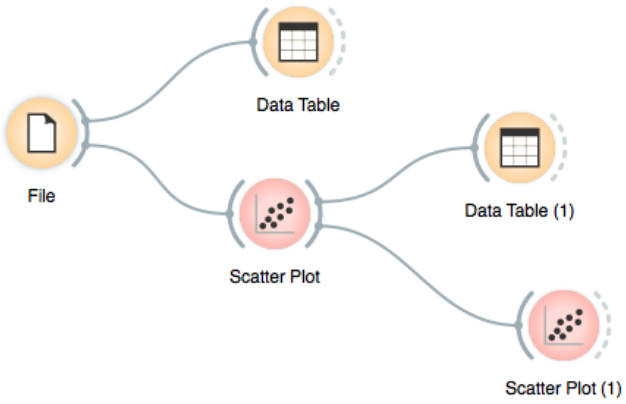
Orange workflows consist of components that read, process and visualize the data. We call them "widgets". Widgets are placed on a drawing board (the "canvas"). Widgets communicate by sending information along a communication channel. Output from one widget is used as input to another.



A simple workflow with two connected widgets and one widget without connections. The outputs of a widget appear on the right, while the inputs appear on the left.

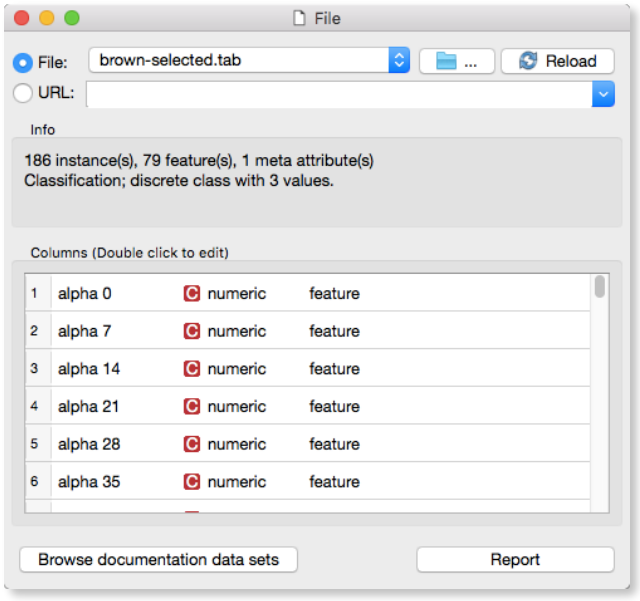
We construct workflows by dragging widgets onto the canvas and connect them by drawing a line from the transmitting widget to the receiving widget. The widget's outputs are on the right and the inputs on the left. In the workflow above, the File widget sends data to the Data Table widget.

Start by constructing a workflow that consists of a *File* widget, two *Scatter Plot* widgets, and two *Data Table* widgets:



Workflow with a File widget that reads the data from disk and sends it to the Scatter Plot and the Data Table widgets. The Data Table renders the data in a spreadsheet, while the Scatter Plot visualizes it. Selected data points from the Scatterplot are sent to two other widgets: Data Table (1) and Scatter Plot (1).

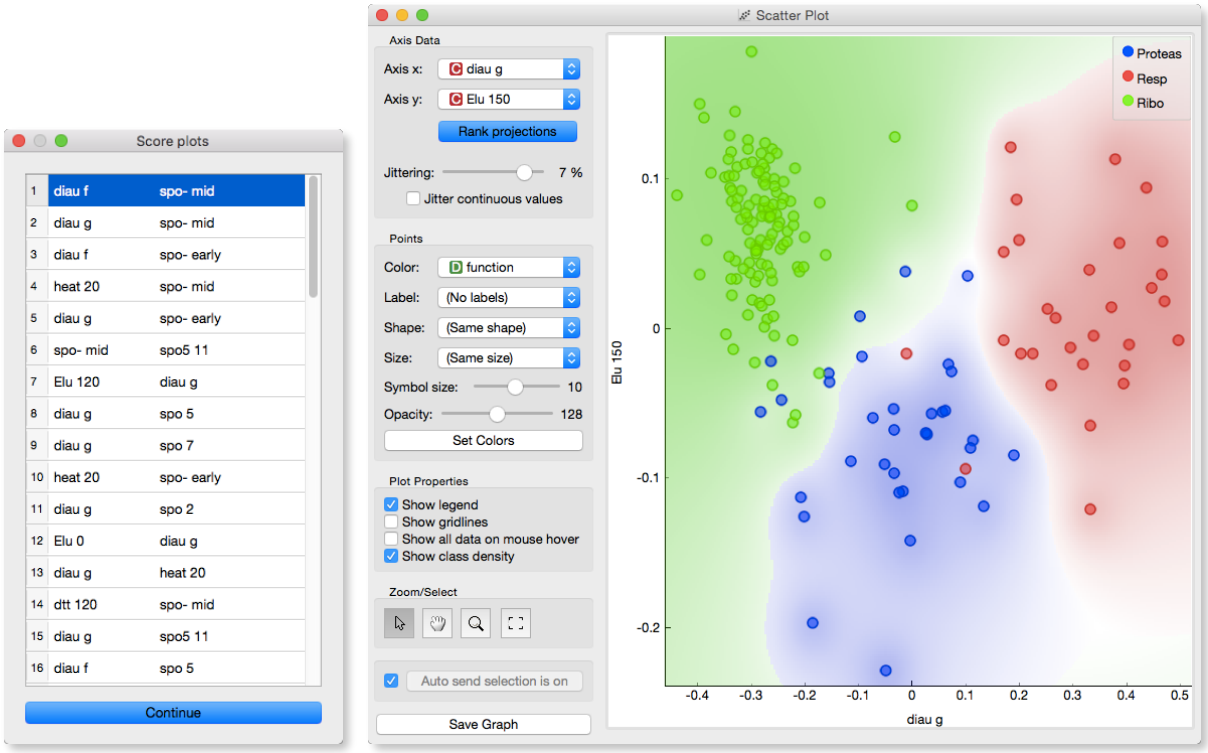
The File widget reads data from your local disk. Open the File widget by double-clicking its icon. Orange comes with several preloaded datasets. From these ("Browse documentation data sets..."), choose *brown-selected.tab*, a yeast gene expression dataset.



Orange workflows often start with a File widget. The brown-selected dataset comprises of 186 rows (genes) and 81 columns. Out of the 81 columns, 79 contain gene expressions of baker's yeast under various conditions, one column (marked as a "meta attribute") provides gene names, and one column contains the "class" value or gene function.

After you load the data, open the other widgets. In the Scatter Plot widget, select a few data points and watch as they appear in the widget Data Table (1). Use a combination of two Scatter Plot widgets, where the second scatter plot shows a detail from a smaller region selected in the first scatter plot.

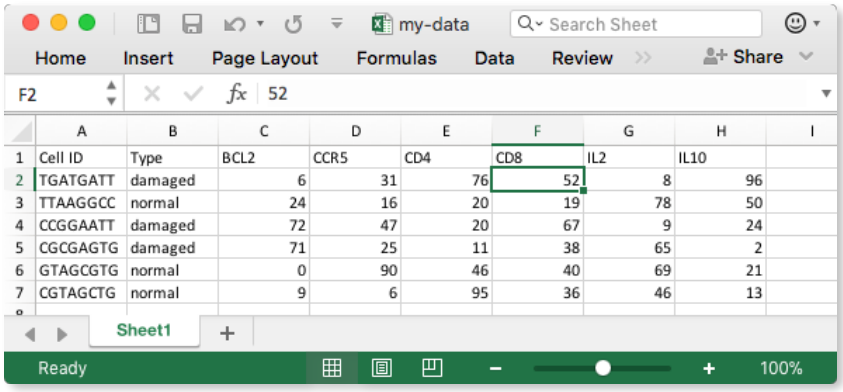
Following is more of a side note, but it won't hurt. Namely, the scatter plot for a pair of random features does not provide much information on gene function. Does this change with a different choice of feature pairs in the visualization? Rank projections (the button on the top left of the Scatter Plot widget) can help you find a good feature pair. How do you think this works? Could the suggested pairs of features be useful to a biologist?



Lesson 2: Loading Your Own Dataset

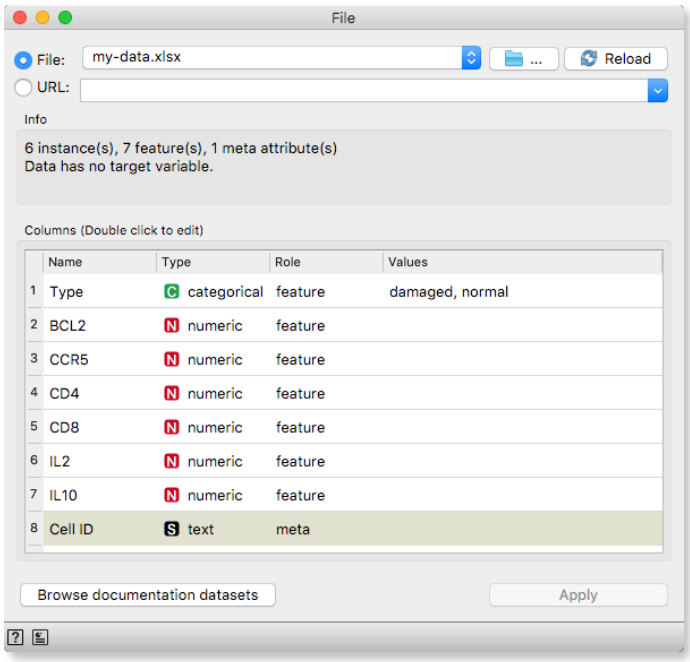
Orange reads data from Excel, comma- and tab-separated files and urls. Try constructing a spreadsheet in Excel or in Google Sheets. If using Google Sheets, copy a shareable link and paste it into the File widget. Press Enter to load the data.

The datasets we have worked with in the previous lesson come with Orange installation. Orange can read the data from spreadsheet file formats which include tab and comma separated and Excel files. Let us prepare a toy dataset in Excel and save it on a local disk.



	A	B	C	D	E	F	G	H	I
1	Cell ID	Type	BCL2	CCR5	CD4	CD8	IL2	IL10	
2	TGATGATT	damaged	6	31	76	52	8	96	
3	TTAAGGCC	normal	24	16	20	19	78	50	
4	CCGGAATT	damaged	72	47	20	67	9	24	
5	CGCGAGTG	damaged	71	25	11	38	65	2	
6	GTAGCGTG	normal	0	90	46	40	69	21	
7	CGTAGCTG	normal	9	6	95	36	46	13	

In Orange, we can use the File widget to load this dataset.

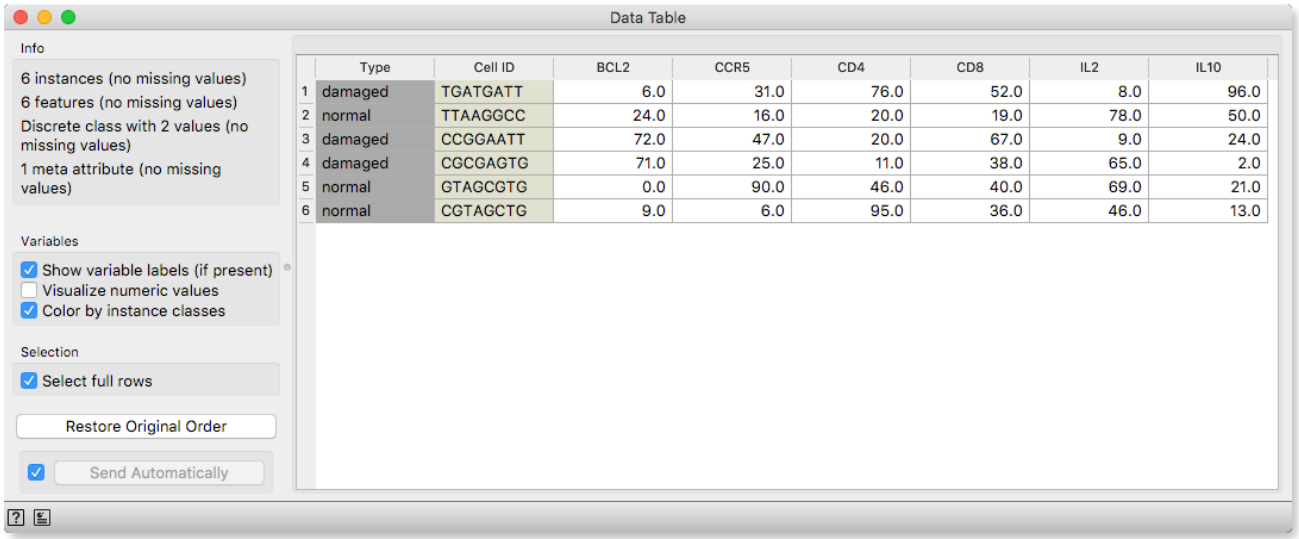


Looks good. Orange has correctly guessed that cell IDs are character strings and that this column in the dataset is special, meant to provide additional information and not to be used for any

kind of modeling. All other columns are numeric features except for the type, which is a categorical feature. This is also the feature we would not like to include in the profile of the cell and should rather consider it as a cell's class. Double-click on the "feature" in the *Role* column and change the role of the feature type to "target". Then click the *Apply* button.

It is always good to check if all the data was read correctly. We can connect our *File* widget with the *Data Table* widget,

and double-click on the *Data Table* to see the data in the spreadsheet format.



	Type	Cell ID	BCL2	CCR5	CD4	CD8	IL2	IL10
1	damaged	TGATGATT	6.0	31.0	76.0	52.0	8.0	96.0
2	normal	TTAAGGCC	24.0	16.0	20.0	19.0	78.0	50.0
3	damaged	CCGGAATT	72.0	47.0	20.0	67.0	9.0	24.0
4	damaged	CGCGAGTG	71.0	25.0	11.0	38.0	65.0	2.0
5	normal	GTAGCGTG	0.0	90.0	46.0	40.0	69.0	21.0
6	normal	CGTAGCTG	9.0	6.0	95.0	36.0	46.0	13.0

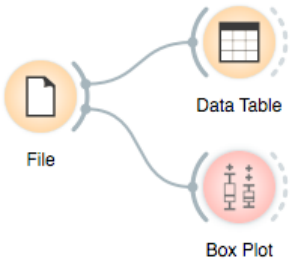
Nice, everything is here.

There is more to input data formatting and loading. We can define the type and kind of the data column, specify that the column is actually a web address of an image, and more. But enough for now. If you would really like to dive in for more, check out the documentation page on Loading your Data, or a video on this subject.

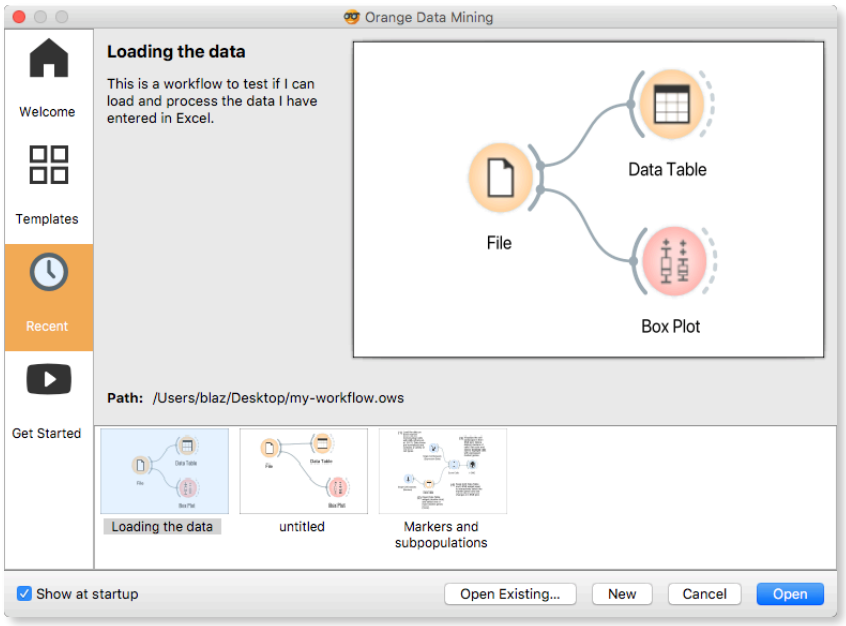
Instead of using Excel, we could also use Google Sheets, a free online spreadsheet alternative. Then, instead of finding the file on the local disk, we would enter its URL address to the File widget 's URL entry box.

Lesson 3: Saving and Sharing Your Work

Add a box plot to your workflow from the previous lesson. Your workflow should now look like this:



Orange can save your workflow to a file. Use Save or Save As from the File menu and save your work to your desktop. Before saving, you can describe your workflow on the information page using "i" icon for the toolbar. We have named the file as *my-workflow* and Orange would add a suffix to it. Now exit Orange and rerun it. Select *Recent* in the welcome screen and choose from one of the recently saved workflows.

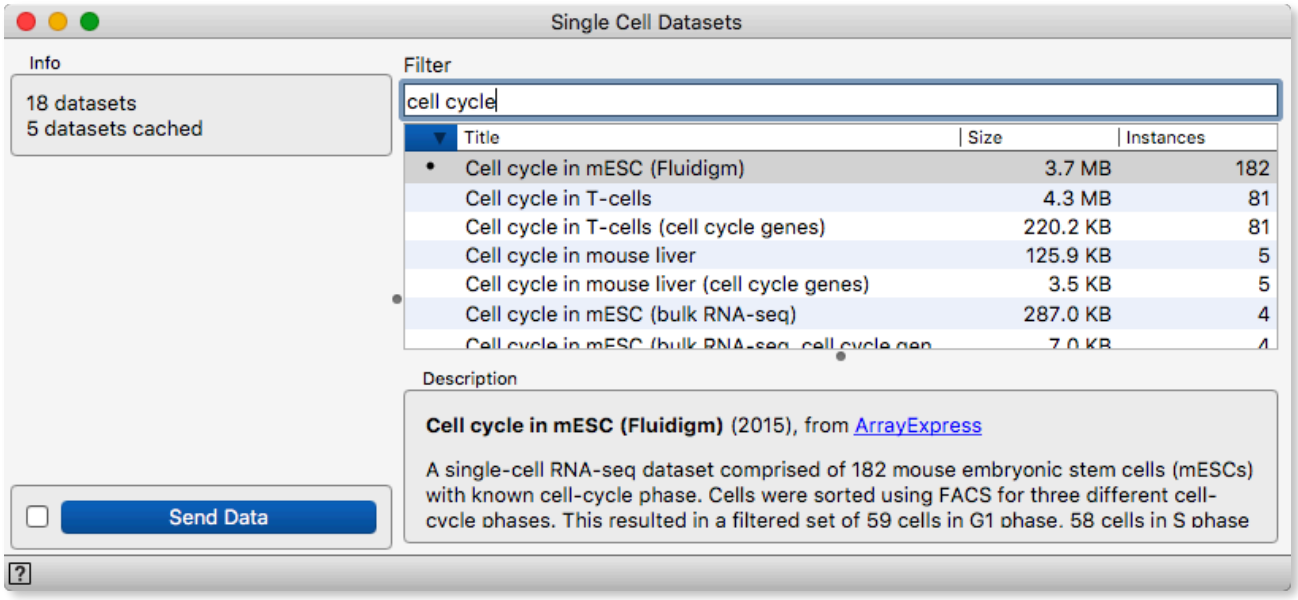


You can email workflow files or share them with your colleagues. Note though that the data, unless stored on the web, has to be sent separately.

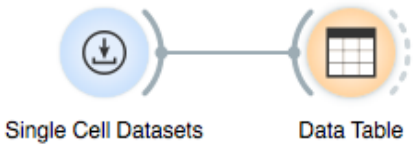
Lesson 4: Basic Exploration of Single Cell Data

Single-Cell Orange (scOrange) provides a quick-start interface to several recent datasets from single-cell RNA publications. In this lesson, we will look at gene expression changes in the cell cycle stages of mouse embryonic stem cells.

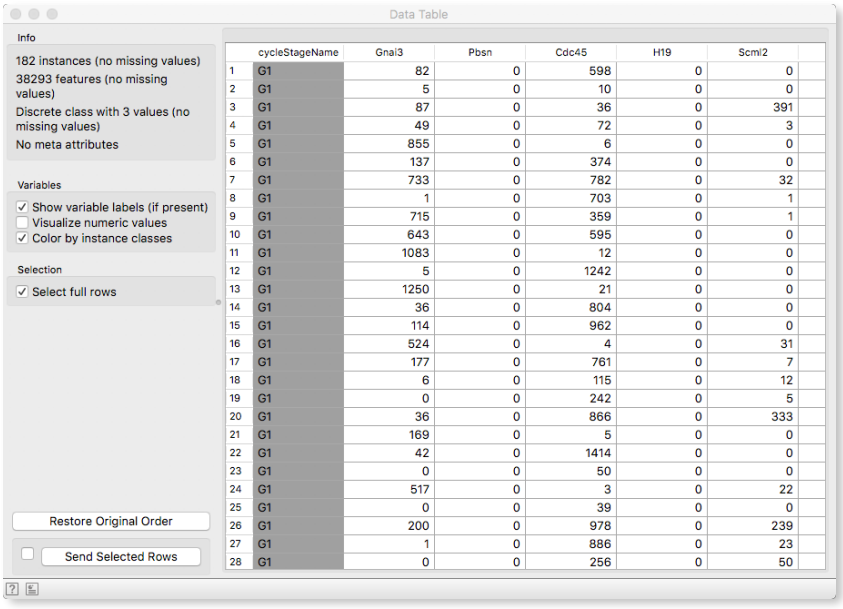
Use the *Single Cell Datasets* widget and load the *Cell cycle in mESC (Fluidigm)* dataset with 182 cells and a full set of 38,293 genes.



You can get a first glance of the data — as before — with the *Data Table* widget.

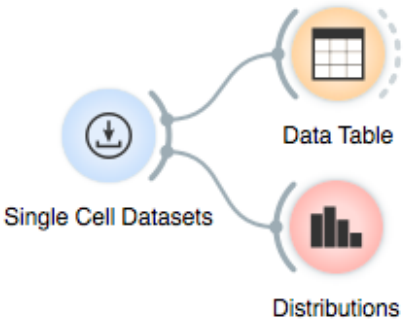


Single cell data can use different quantification measures, like read counts or transcripts counts. Our cell cycle data includes read counts.

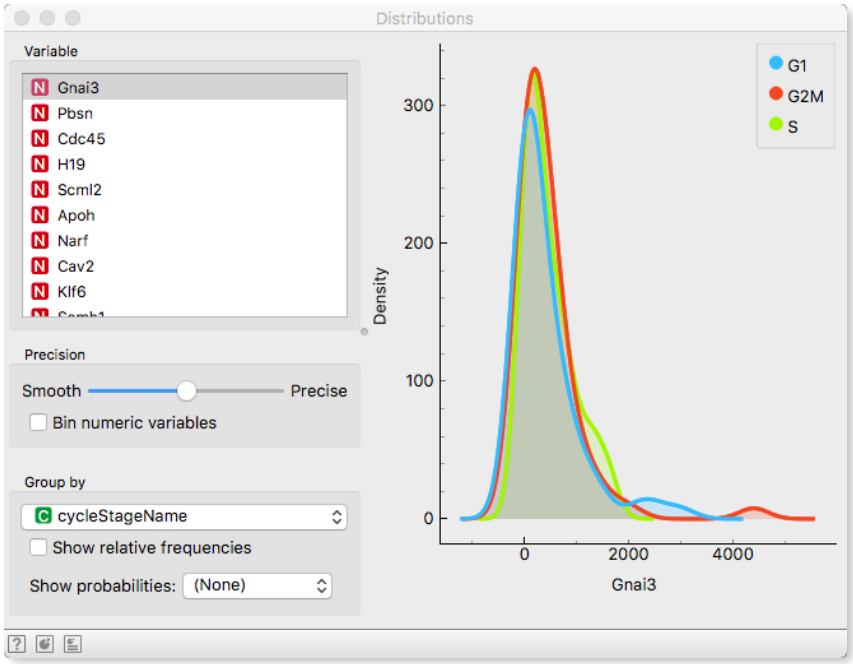


The program scOrange stores the data tables with cells in rows and genes in columns. For each cell, we have the count of sequence reads associated with each of the vast numbers of genes. In other words, we have a direct quantification of gene expression profiles. There are a few things to notice here — what can we say about the range of counts stored in the data table?

The counts vary wildly across cells, but also across genes — this is a typical example of technical variance that is compensated with downstream statistical methods. We can visualize the distribution of counts with the *Distribution* widget. Select different genes and see how genes generally either display very heavy-tailed distributions of counts or are not detected at all.



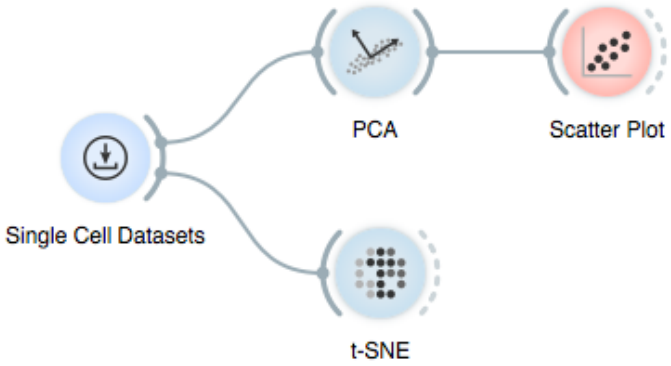
Set the *Group By* property to *cycleStageName* to see the distribution of counts for each of the three subgroups of cells.



Heavy-tailed and sparse data does not play well with conventional data analysis techniques. In the next section, we will do something about this.

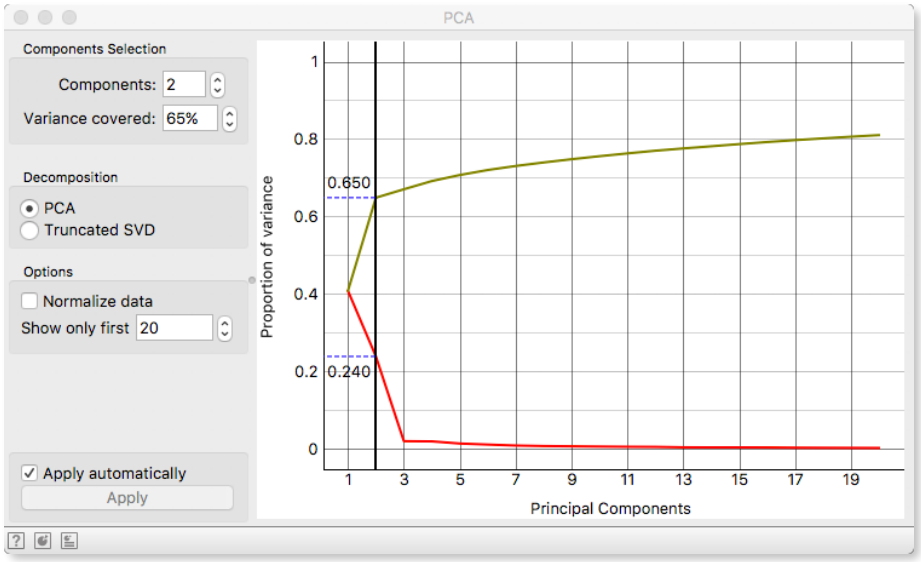
But first, a short detour.

Lesson 5: Data Projection with PCA and t-SNE



Looking at the values in a large table will probably not get us very far — except if you're a fan of a certain spreadsheet program. We can instead start with the *Principal component analysis (PCA),* which is a way to quantify variance of different directions in multi-dimensional data. Selecting directions with the highest explained variance is one way to reduce the dataset size. If we consider only two directions — the two principal components — we can visualize the data in a scatter plot.

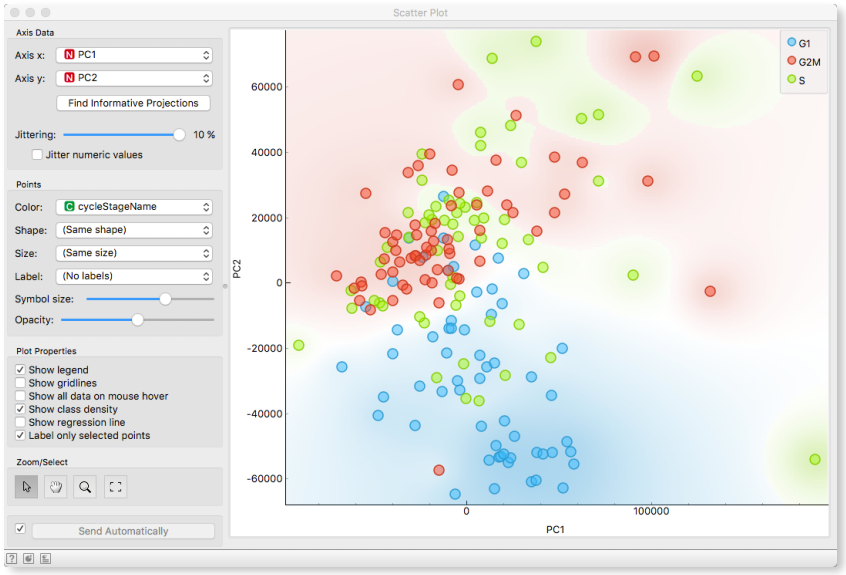
Note that 'Normalized data' checkbox is ticked off in the PCA; simply standardizing the gene expressions would amount to some loss of information on relative expression levels among genes.



The first two components explain a gigantic 65% of the variance, which is great, considering that we are dealing with 182-dimensional data!

However, the large magnitude on the axes reminds us of the very heavy-tailed distributions and raises an alarm that the projection might be dominated by a small number of high count values. We are postponing this problem to the next section.

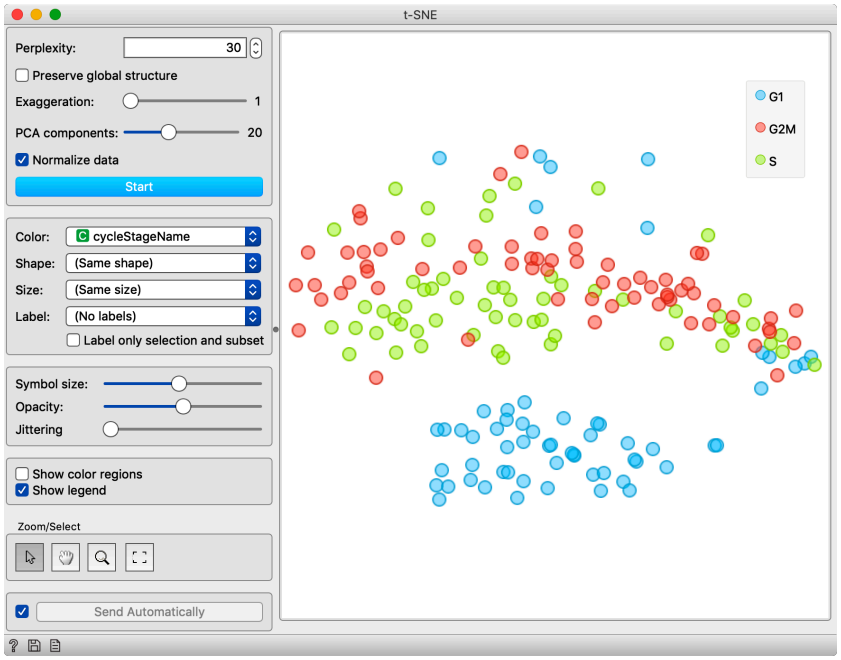
A word of caution: t-SNE essentially discards the true distances between the cells. Could the 2-D t-SNE coordinates be used in further downstream analyses, e.g. finding clusters?



The PCA + Scatter plot combo shows a mixed result: there is substantial overlap between cells in a different cell-cycle stage.

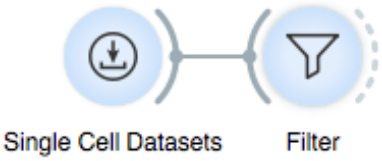
While PCA is a linear projection, the *t-distributed Stochastic Neighbour Embedding (t-SNE)* is a technique that is more suited to how humans interpret 2-D projections. With t-SNE, the main goal is to project very similar cells close together, while the distances between dissimilar cells do not significantly influence the positioning. This relaxation prevents spherical structures as those commonly seen in PCA, where the distances between points are treated equally. Hence t-SNE can yield more clustered, grape-like structures.

Try to modify the *Perplexity* parameter of the t-SNE plot. How does this value influence the visualization? What kind of values are more suitable to detect local structure and what for global structure?



Lesson 6: Data Filtering

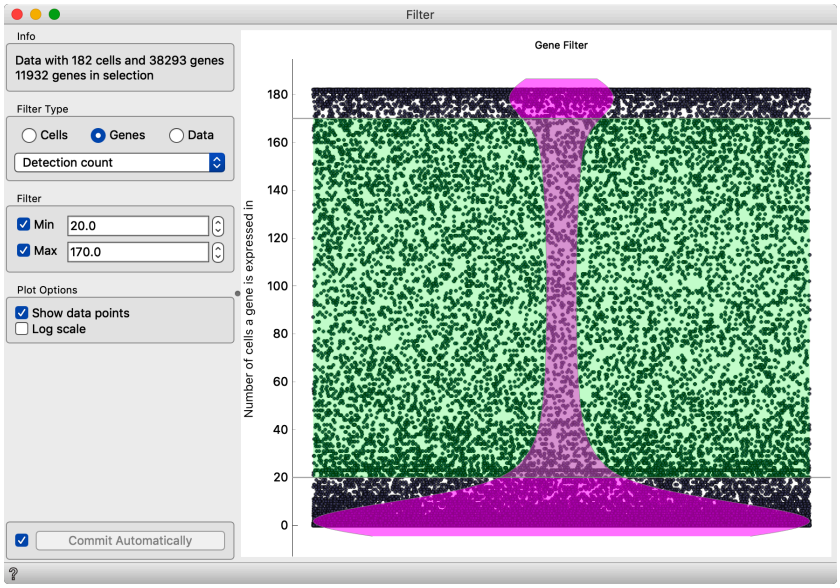
As we have previously seen, single cell data is an expression matrix with tens of thousands of genes and hundreds or thousands of cells. A natural question to ask is whether all of the data is required for further analysis.



Using the *Filter* widget, we can retain only the genes that are expressed in a certain minimal or maximal number of cells. For this example, let's set the minimal/maximal threshold to 20/170 cells (out of 182), respectively. From a total of 38,293 genes, we retain 11,932 genes. In other words, we can reduce the dataset size by 75%!

By setting a lower bound on the number of cells per gene, we remove the genes that are unlikely to significantly influence the analysis. By removing the very highly expressed genes, the *housekeeping* (ever-present) genes are removed.

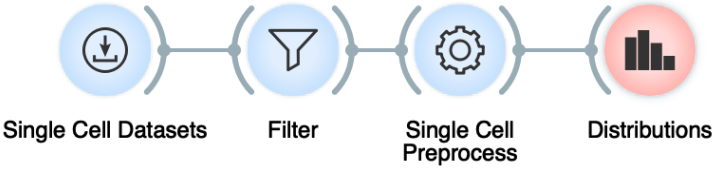
A similar filtering can be performed for cells by setting the lower/upper bound on the number of detected genes.

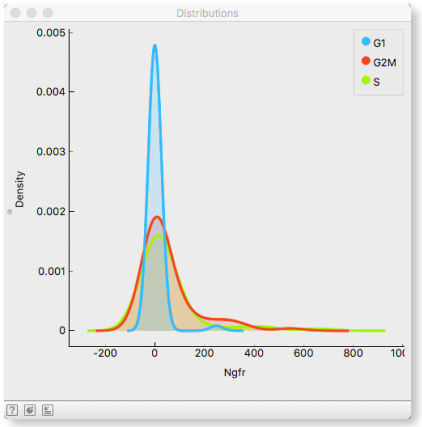


Lesson 7: Normalization

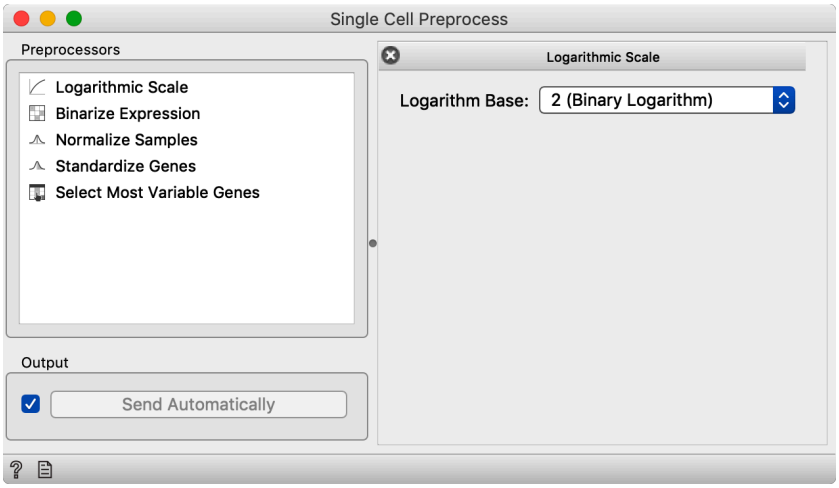
We can also tackle the technical variability issues. Due to a non-uniform distribution of reads per cell, some cells will *a priori* have larger counts associated to genes. A simple form of normalization is achieved when the gene count profiles are scaled to have the *same median value for all cells*. Alternatively, the *same median* constraint can be imposed on groups of cells which in this case can be defined as cells in the same cell cycle phase. We perform normalization after the data filtering.

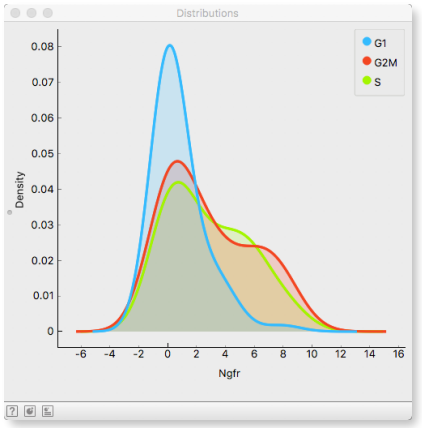
You can also try to use *PCA* widget in the last step. How do the coordinate axes change after preprocessing?





The distributions show the expression of gene *Ngfr* before and after normalization. Notice how we got rid of the long tail!



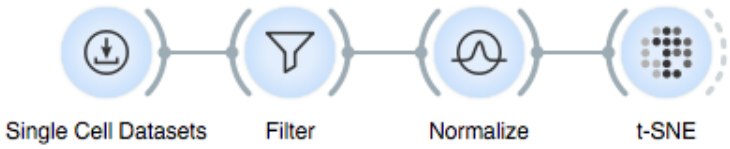


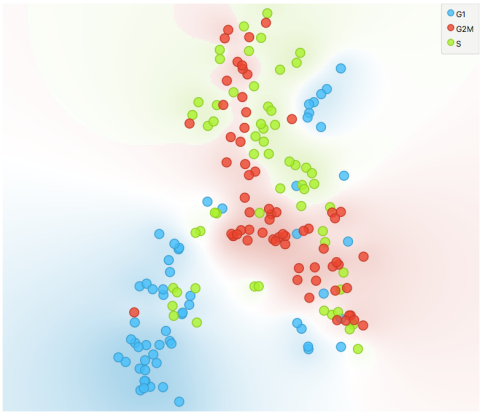
By scaling the cell expression profiles by a factor does not get rid of the heavy-tailed gene expression distributions. To achieve this, we use the following transformation:

$$y = \log(x + 1)$$

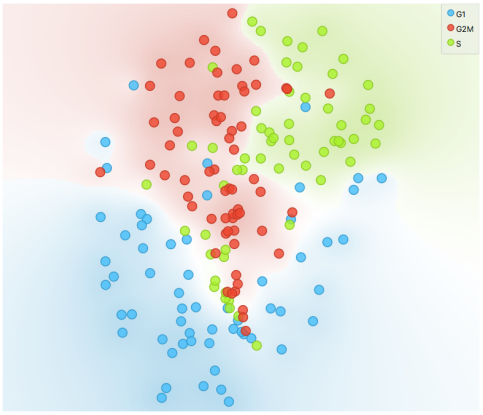
Why does this transformation make sense? What happens to genes with an expression equal to zero in the raw and transformed data? Try using *Distribution* widget to obtain the answer.

To get a feeling of what the filtering and normalization do to the data, let's create a sequence of t-SNE plots after applying each of the preprocessing steps. The preprocessing does quite a nice job, focusing on the genes that might be important for separating cells in different phases of the cell cycle!

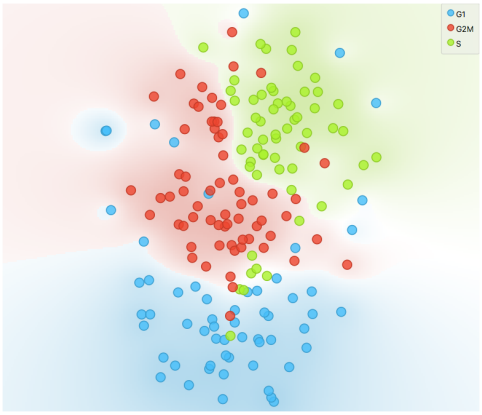




Raw data



Filtered

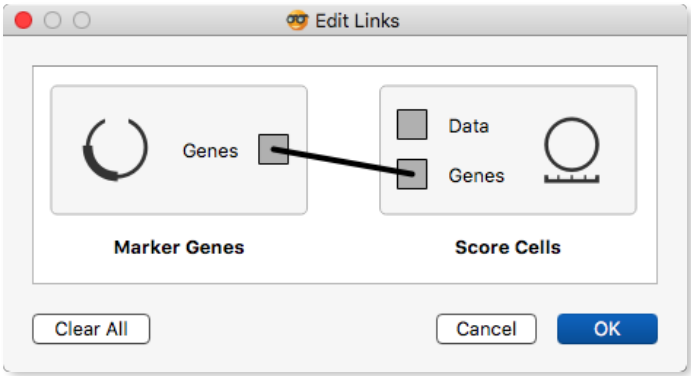


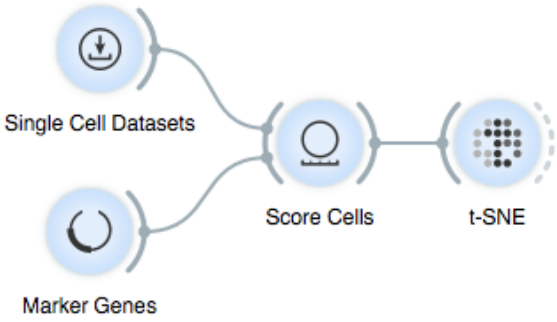
Filtered + normalized

Lesson 8: Marker Genes

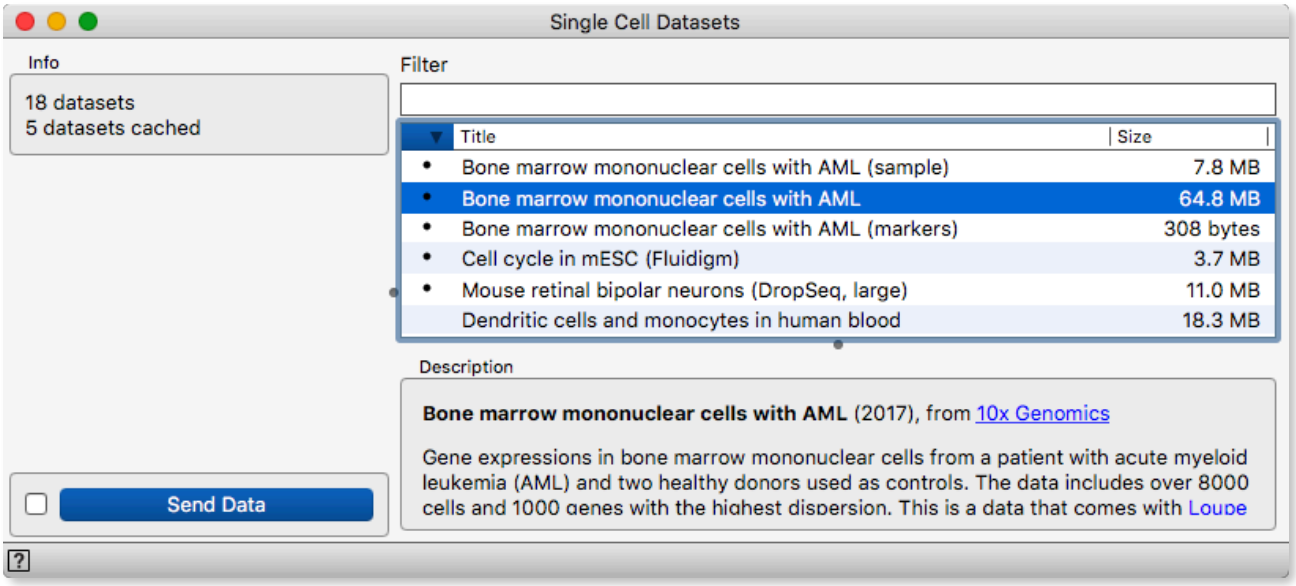
Score Cells widget has two inputs: the data and a set of marker genes. Since both inputs are data tables, how does it know which one is which? Double-click on any of the edges connecting the widgets to see the actual wiring. You can rewire the connections there.

Gene markers are one of the common methods for discovering cell populations, determining cell states (e.g. cell cycle) and more. Here, we will use a sample of the data on bone marrow mononuclear cells with AML (Zheng et al., Nat Comm 2017), score the cells according to selected gene markers, and observe the scored cells in the t-SNE plot:

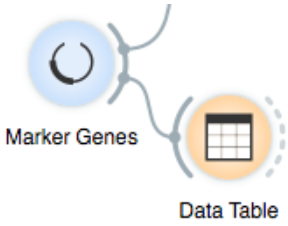




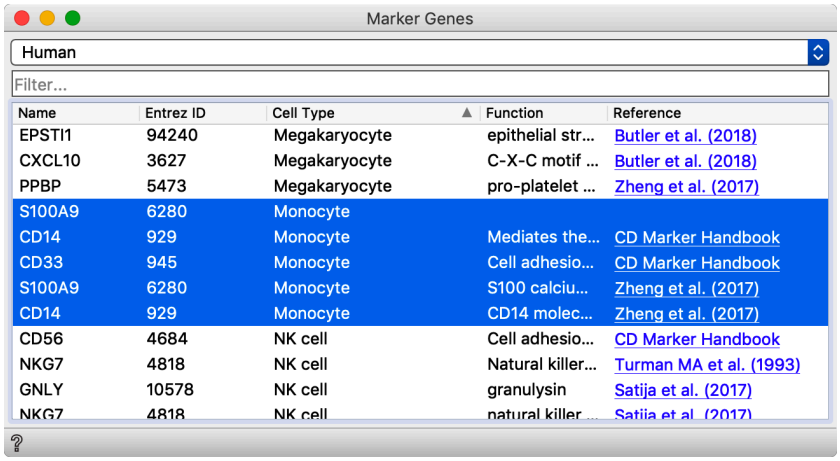
The sample bone marrow dataset from *Single Cell Datasets* widget includes 1000 genes and 1000 cells. There is a bigger dataset available from the same widget (8000 cells), but let us continue with the sample and define our workflow first. Then, you are free to choose larger datasets.



A widget *Marker Genes* offers a list of markers and outputs a data table with selected rows. You can test this by connecting it to *Data*

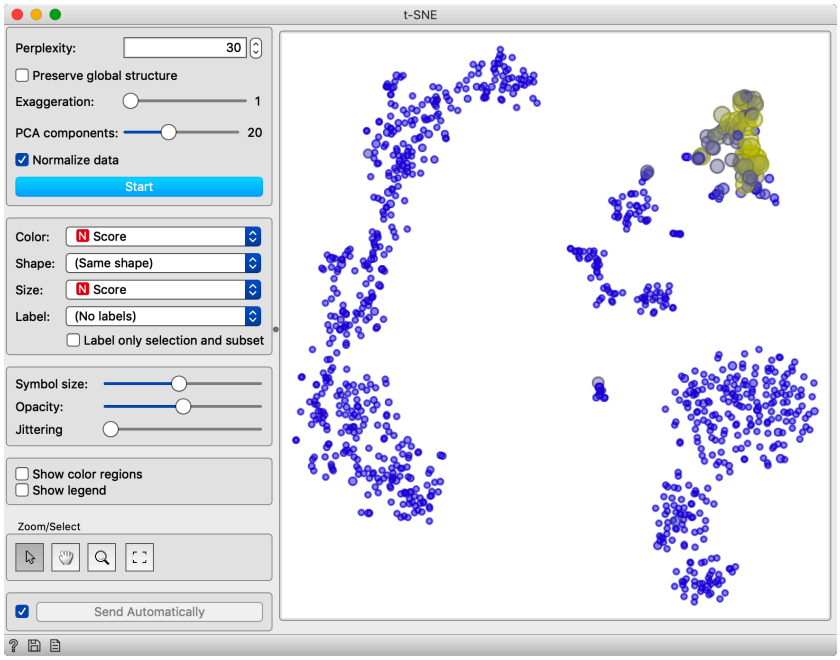


Table, selecting some rows from *Marker Genes* and inspecting the output of this widget in the *Data Table*. We can select one or several marker genes at a time:

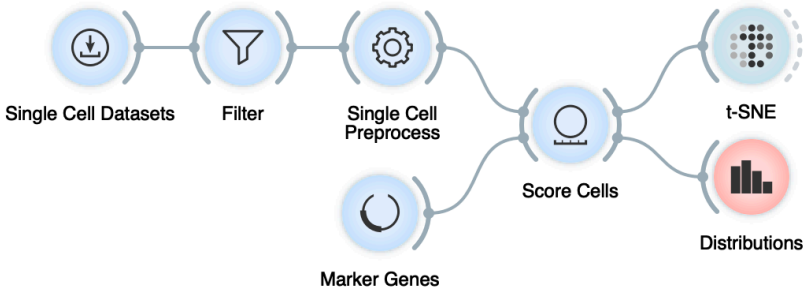


Widget *Score Cells* assigns a numerical score to each cell that is proportional to an average expression of the marker genes at the input of the widget. The score is added as a meta attribute to the cell data on the output of *Score Cells*. Check this using the *Data Table*! We can now feed this data into *t-SNE* and set the color and size of the points to the cell score:

In the t-SNE widget set the point color to cell score. Open the *Marker Genes* widget and change the selected marker. See the changes in the t-SNE widget. We have just assembled a workflow for interactive display of the marker genes!

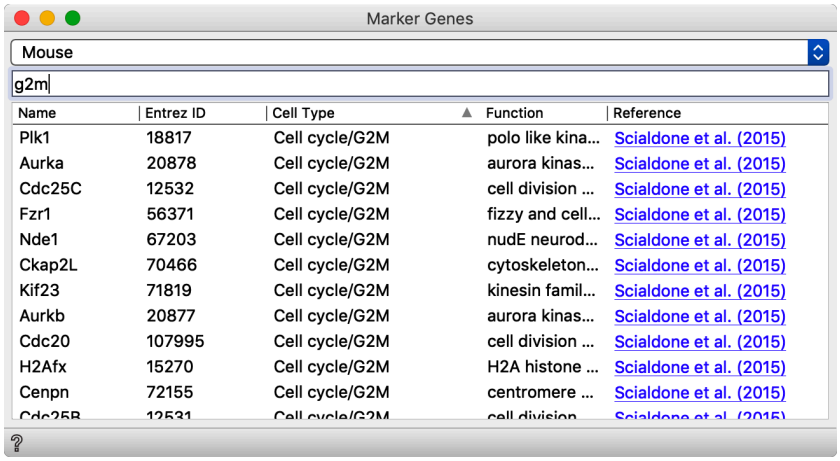


We can now also study the marker genes in the cell cycle data from the previous lesson. Make sure to change the organism in the *Marker Genes* widget to mouse. You will notice that the differences between cells are less pronounced. Use the following workflow:



We would like to compare the scores of the cells for G2M phase markers versus the G1/S, and observe if the distribution of the scores does indeed match the types of cells and markers identified in the original work. Open *Marker Genes* and in the mouse section of the genes select those for G2M (use *Filter* field):

To select all marker genes for a particular cell type, specify the type in the filter entry and press Cmd-A.

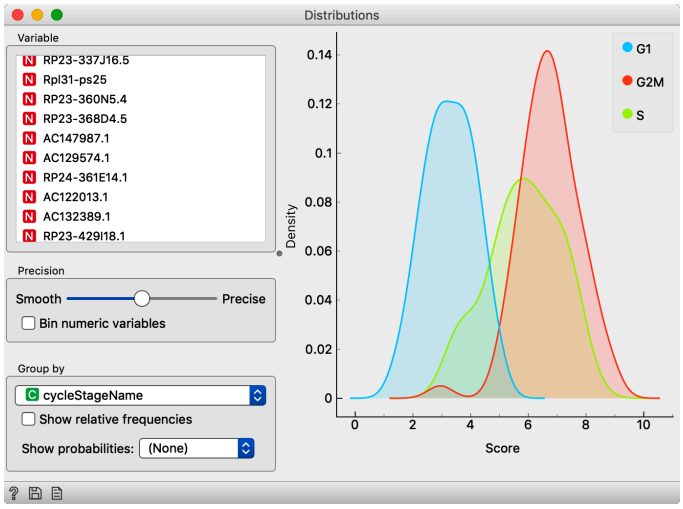


Name	Entrez ID	Cell Type	Function	Reference
Plk1	18817	Cell cycle/G2M	polo like kina...	Scialdone et al. (2015)
Aurka	20878	Cell cycle/G2M	aurora kinas...	Scialdone et al. (2015)
Cdc25C	12532	Cell cycle/G2M	cell division ...	Scialdone et al. (2015)
Fzr1	56371	Cell cycle/G2M	fizzy and cell...	Scialdone et al. (2015)
Nde1	67203	Cell cycle/G2M	nudE neurod...	Scialdone et al. (2015)
Ckap2L	70466	Cell cycle/G2M	cytoskeleton...	Scialdone et al. (2015)
Kif23	71819	Cell cycle/G2M	kinesin famil...	Scialdone et al. (2015)
Aurkb	20877	Cell cycle/G2M	aurora kinas...	Scialdone et al. (2015)
Cdc20	107995	Cell cycle/G2M	cell division ...	Scialdone et al. (2015)
H2Afx	15270	Cell cycle/G2M	H2A histone ...	Scialdone et al. (2015)
Cenpn	72155	Cell cycle/G2M	centromere ...	Scialdone et al. (2015)
Cdc25B	12531	Cell cycle/G2M	cell division	Scialdone et al. (2015)

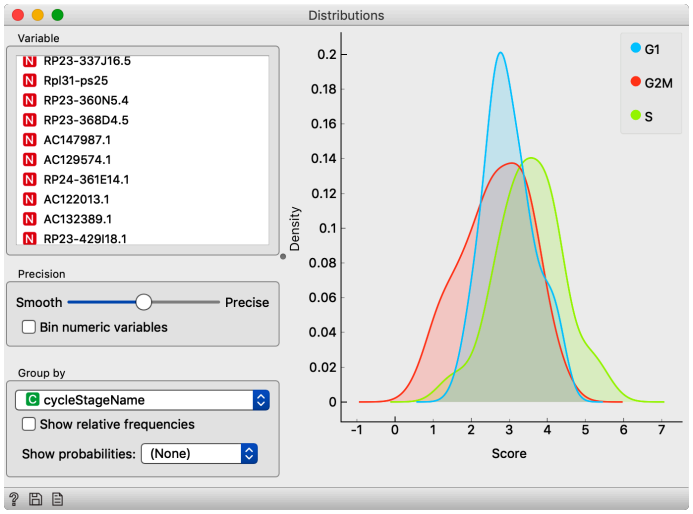
The score distributions for different types of cells indeed confirm the role of the markers: there is a difference in the mean expression of the G2M phase markers versus the G1/S.

We can confirm that the markers indeed differentiate between cells in the three different cycles. The differences, though, are not very substantial.

Below is the distribution we obtain for G2M markers.



The following is a distribution we obtain for G1/S markers:



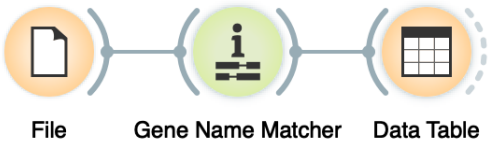
On a single gene level, the difference between cell cycle phases is very narrow but still visible. In the next section, we will introduce methods for finding combinations of genes that will hopefully give use a more clear separation of genes into different classes — cell cycle stages.

Lesson 9: Your Own Markers

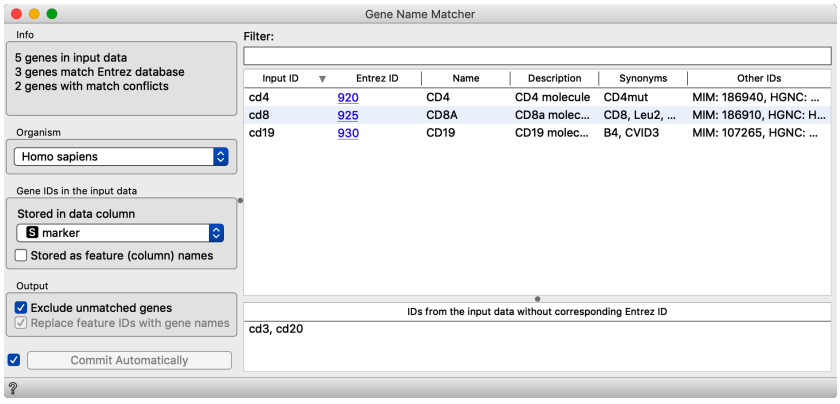
The markers available in Marker Genes widgets were collected from some publications and are by no means complete. Researchers often use their own set of markers, either collected from the literature or hypothesized on from their own research.

In this lesson, we will use Excel or some other spreadsheet program to define our own set of markers for the AML data set. Find "CD Marker Handbook" that is freely available on the internet, use a spreadsheet program to enter the names of the genes and their types, and use them in the Orange workflow. Your data set could look something like the following:

marker	type
cd3	T Cell
cd4	T Cell
cd8	T Cell
cd19	B Cell
cd20	B Cell

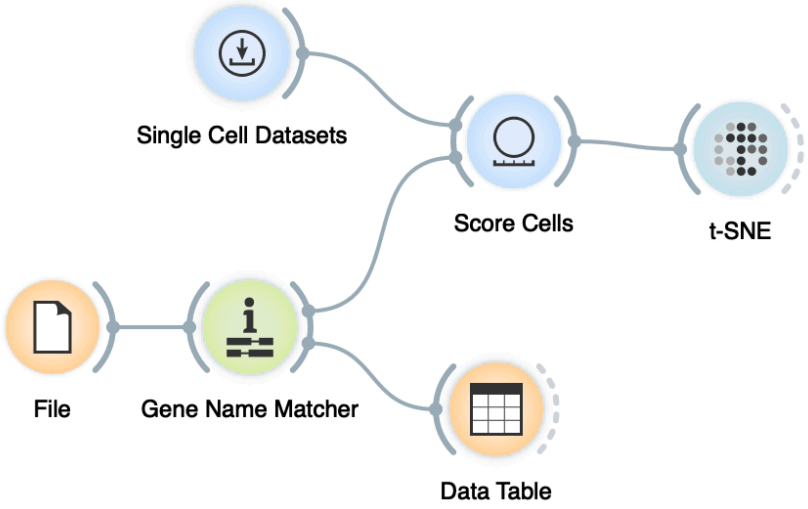


This time, use the File widget to open the data with a list of markers. We need to convert the names of the genes to the standard ID's, and should use Gene Name Matcher widget:



Have all the names of the genes been successfully converted to the IDs? Orange uses NCBI's data bases for gene name synonyms, and it happens that the name, like cd3, is actually an alias for several genes, hence matching fails. Check the result of the matching in the Data Table.

Your final workflow with loader of your marker genes, cell scorer, and t-SNE visualization of scores should not look like the following:

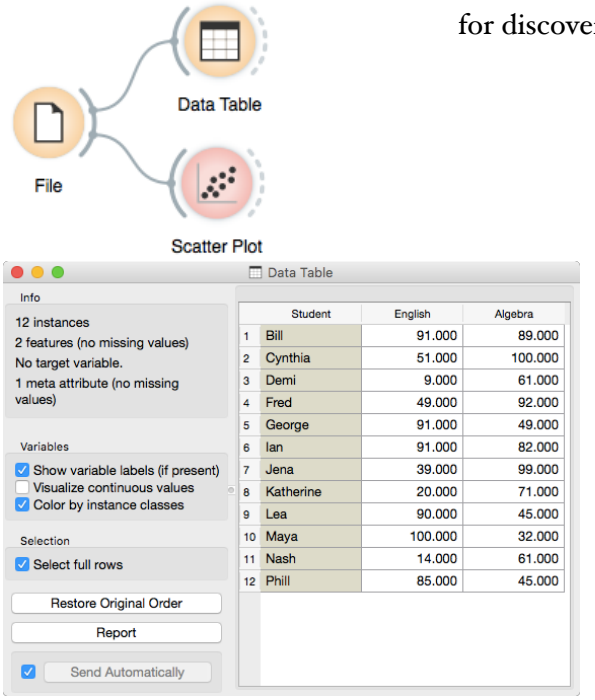


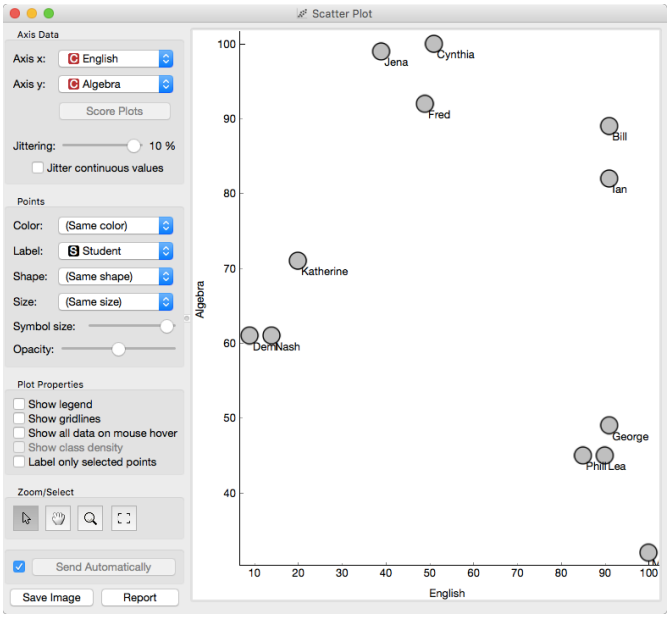
Try choosing various genes or gene sets in the Gene Name Matcher. You could also connect the output of the Data Table to the Score Cells to provide the selection of the genes.

In the class, we will introduce clustering using a simple data set on students and their grades in English and Algebra. Load the data set from http://file.biolab.si/files/grades2.tab.

Lesson 10: Hierarchical Clustering

Slight deviation here. We will use hierarchical clustering on single cell data a bit later, but let us learn about this clustering method first. We would like to identify groups of data instances (cells) that are close together, similar to each other. Consider a simple, two-featured data set (see the side note) and plot it in the Scatter Plot. How many clusters do we have? What defines a cluster? Which data instances belong to the same cluster? What would a procedure for discovering clusters look like?





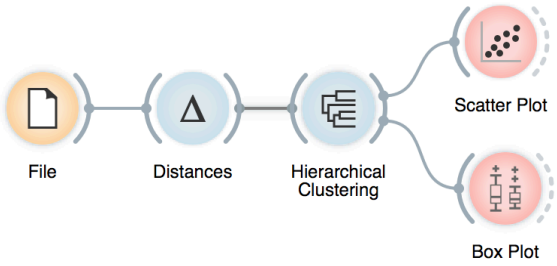
How do we measure the similarity between clusters if we only know the similarities between points? By default, Orange computes the average distance between all their pairs of data points; this is called average linkage. We could instead take the distance between the two closest points in each cluster (single linkage), or the two points that are furthest away (complete linkage).

We need to start with a definition of "similar". One simple measure of similarity for such data is the Euclidean distance: square the differences across every dimension, some them and take the square root, just like in Pythagorean theorem. So, we would like to group data instances with small Euclidean distances.

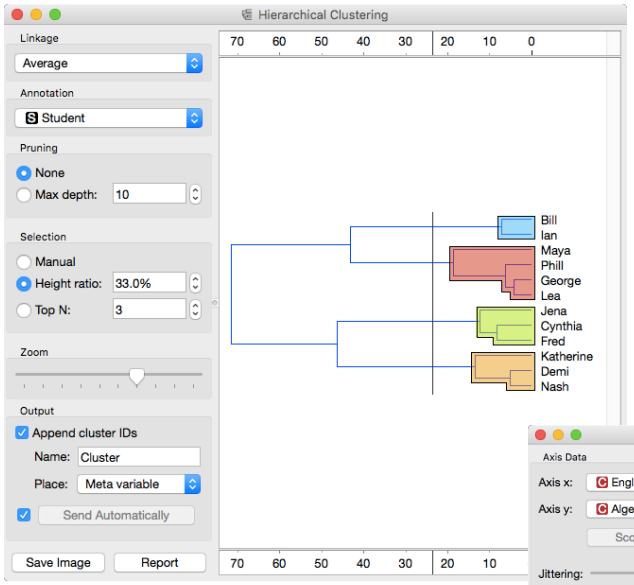
Now we need to define a clustering algorithm. We will start with each data instance being in its own cluster. Next, we merge the clusters that are closest together - like the closest two points - into one cluster. Repeat. And repeat. And repeat. And repeat until you end up with a single cluster containing all points.

This procedure constructs a hierarchy of clusters, which explains why we call it hierarchical clustering. After it is done, we can observe the entire hierarchy and decide which would be a good point to stop. With this we decide the actual number of clusters.

One possible way to observe the results of clustering on our small data set with grades is through the following workflow:



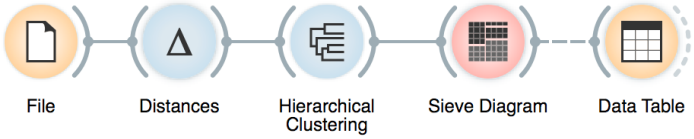
Let us see how this works. Load the data, compute the distances and cluster the data. In the Hierarchical clustering widget, cut hierarchy at a certain distance score and observe the corresponding clusters in the Scatter plot.



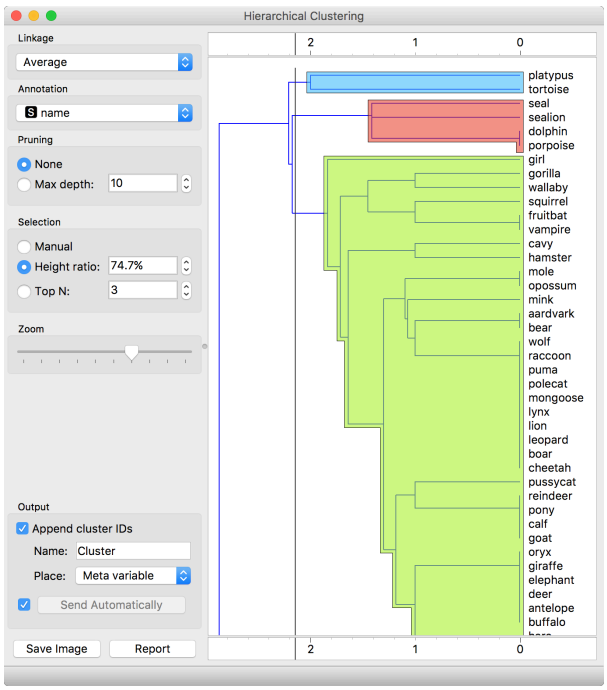
You can also observe the properties of the clusters - that is, the average grades in Algebra and English - in the box plot.

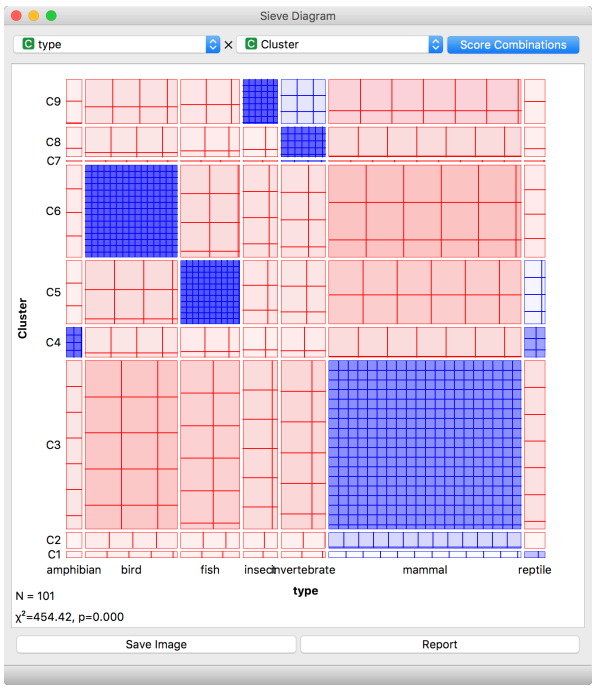
Lesson 11: Animal Kingdom

Your lecturers spent substantial part of their youth admiring a particular Croatian chocolate called Animal Kingdom. Each chocolate bar came with a card — a drawing of some (random) animal, and the associated album made us eat a lot of chocolate. Then our kids came, and the story repeated. Some things stay forever. Funny stuff was we never understood the order in which the cards were laid out in the album. We later learned about taxonomy, but being more inclined to engineering we never mastered learning it in our biology classes. Luckily, there's data mining and the idea that taxonomy simply stems from measuring the distance between species.



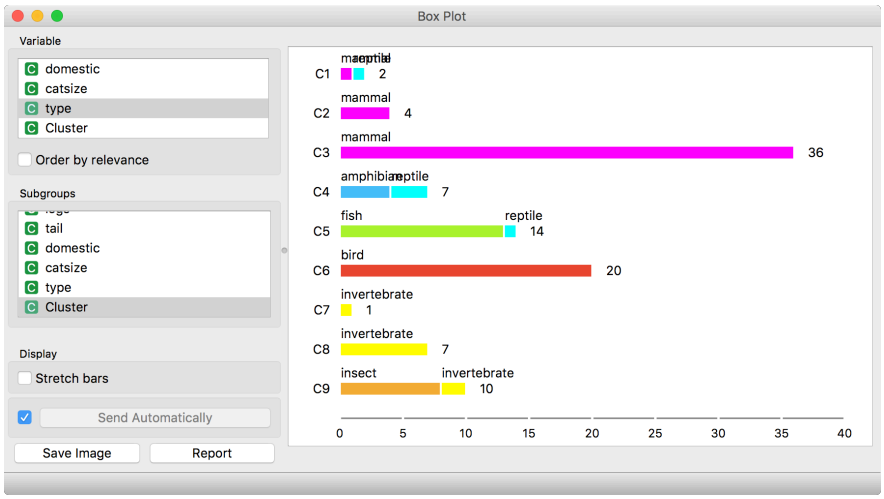
Here we use zoo data (from documentation data sets) with attributes that report on various features of animals (has hair, has feathers, lays eggs). We measure the distance and compute the clustering. Animals in this data set are annotated with type (mammal, insect, bird, and so on). It would be cool to know if the clustering re-discovered these groups of animals. We can do this through marking the clusters in Hierarchical Clustering widget, and then observing the results in the Sieve Diagram.



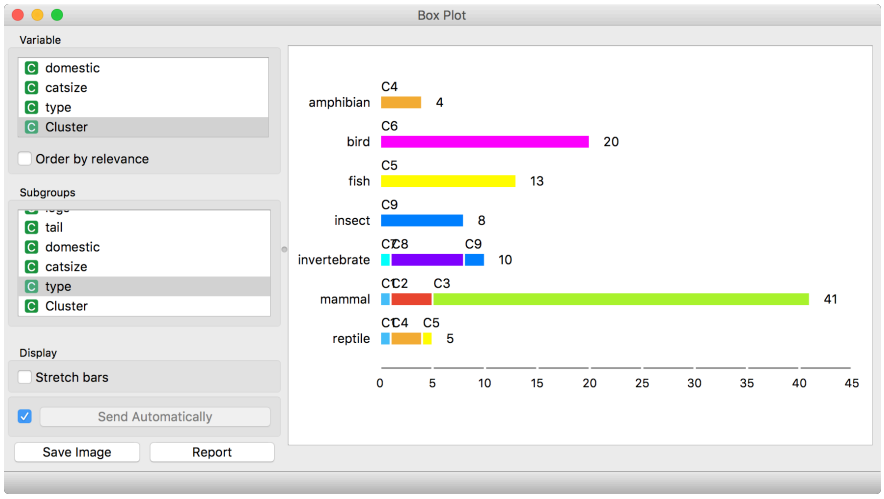


Looks great. Birds, say, are in cluster C6. Cluster C4 consists of amphibians and some reptiles. And so forth.

Checking this in the Box plot is even cooler. We can get a distribution of animal types in each cluster:



Or we can turn it around and see how different types of animals are spread across clusters.

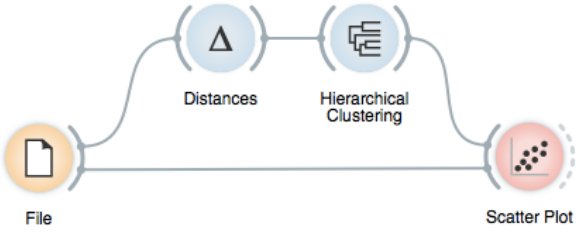


What is wrong with those mammals? Why can't they be in one single cluster? Two reasons. First, they represent 40 % of the data instances. Second, they include some weirdos. Click on the clusters in the box plot and discover who they are.

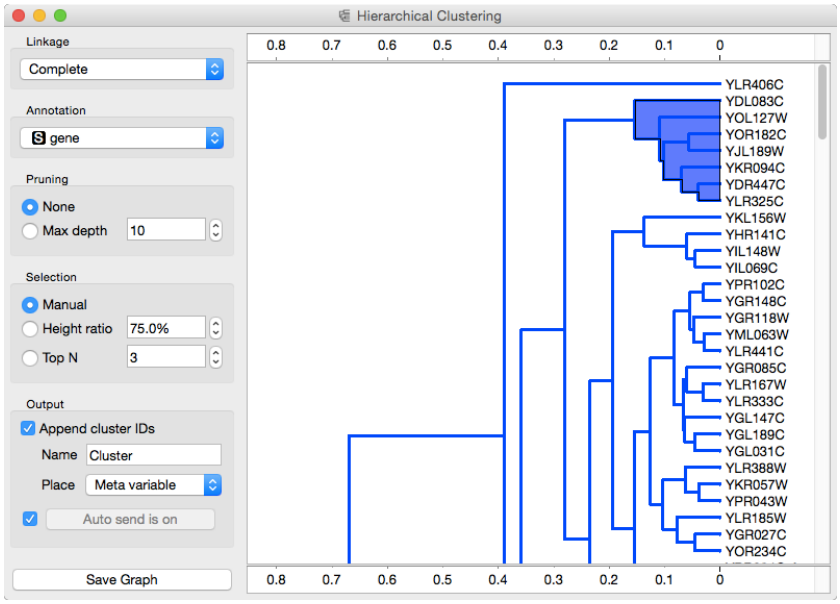
Lesson 12: Discovering clusters

Can we replicate this on some real data? Can clustering indeed be useful for defining meaningful subgroups?

Take brown-selected (from documentation data sets) connect the hierarchical clustering so the you can see a cluster as a subset in the scatterplot.



So far, we used the dendrogram to set a cut-off point. Now we will click on a branch in a dendrogram to select a subset of the data instances. By combining it with the Scatter Plot widget, we get a great tool for exploring the clusters. Try it with an appropriate pair of features to visualize (use Rank projections).



By using a scatter plot or other widgets, an expert can determine whether the clusters are meaningful.

For this data set, though, we can do something even better. The data already contains some predefined groups. Let us check how

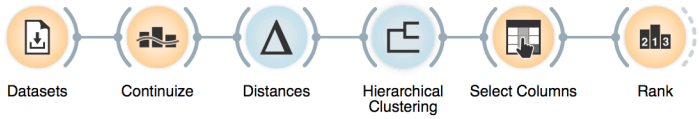
well the clusters match the classes - which we know, but clustering did not.

We will use the dendrogram to set a suitable threshold that splits the data into some three to five clusters. We can plot this data in a new scatter plot; we find a reasonable pair of attributes and then set the color of the points to represent the cluster they belong to. Do the clusters match the actual classes? The result is rather impressive if you keep two things in mind. First, the clustering algorithm did not actually know about the classes, it discovered them by itself. Second, it did not operate on the picture you see in the scatter plot and in which the clusters are quite pronounced, but in a 79-dimensional data space with possibly plenty of redundant features. Yet it identified the three groups of genes almost without mistakes.

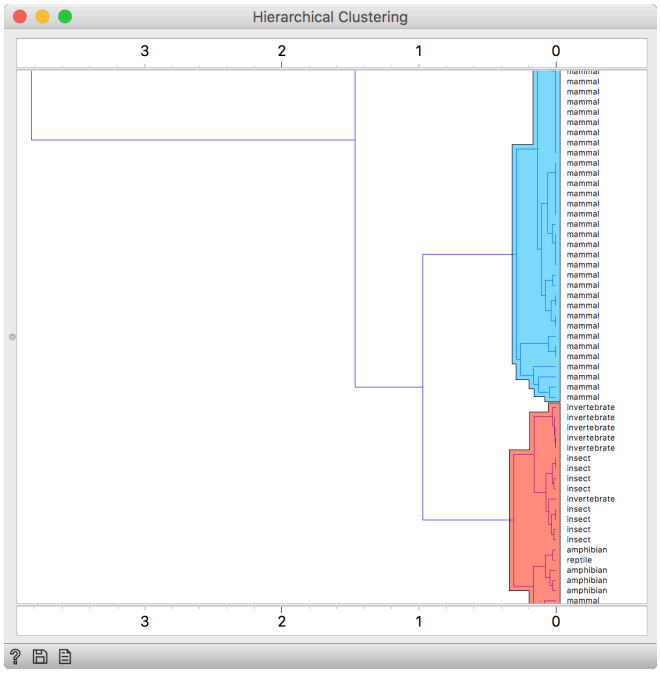
This lessons is not a recipe for what you should be doing in practice. If your data already contains groups labels, say gene group annotations, there is no need to discover them (again) by using clustering. In this case you should be interested in predictive models from previous lessons. If you do not have such a grouping but you suspect that the data contains distinct subgroups, run clustering. The sole purpose of this lesson was to demonstrate that clustering can indeed find a meaningful subgroups in the data; we pretend we did not know the groups, use the clustering to discover them, and checked how well the correspond to the actual groups.

Lesson 13: Cluster Interpretation

Once we have inferred the clusters, we would like to know what are the distinguishing features. For the zoo data set, we could, for instance, mark two clusters, and then ask for the features that distinguish among these. Having data marked with cluster identifiers takes us back to classification, and we can use any of visualization, model inference, or feature ranking techniques we have introduced there. Here, we will show how to use ranking to infer what features characterize the group of mammals when compared to a close cluster of other species.

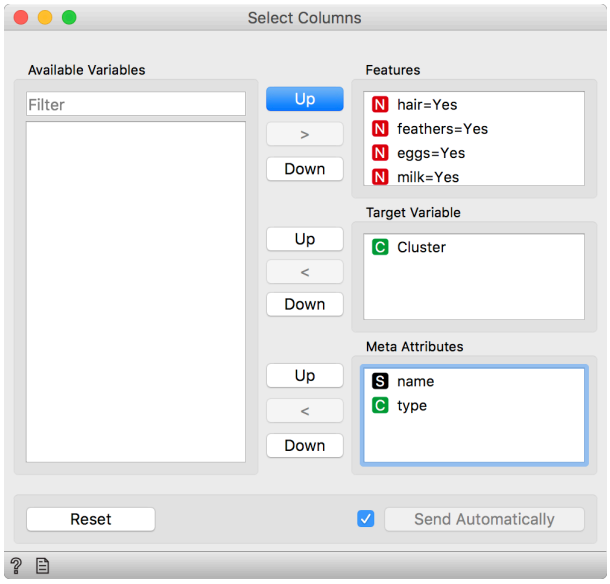


We load zoo data from Datasets, continuize the categorical features (this will be changed in Orange soon, Distances should automatically perform continuization), estimate the distances, and feed everything in Hierarchical Clustering. So far, nothing new.

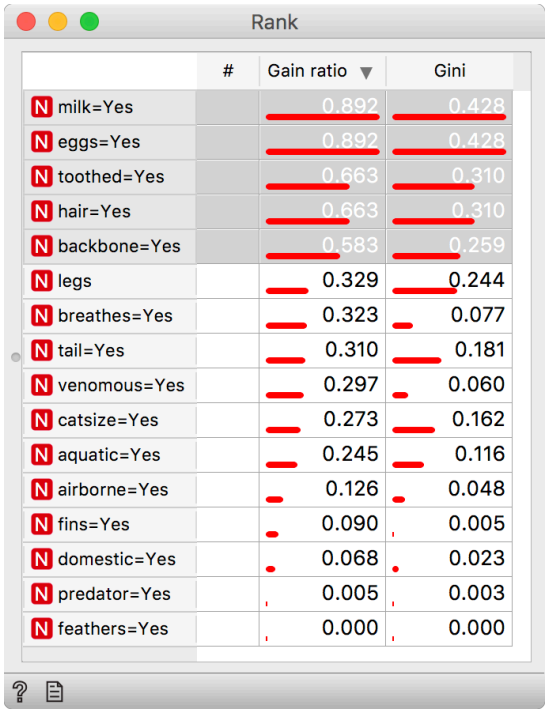


Use modifier keys (command) to select different branches of the dendrogram and mark them as separate clusters. Done correctly, Hierarchical Clustering will mark the branches with different colors.

In the hierarchical clustering, we choose two clusters. Note that Hierarchical Clustering adds cluster identifier as a meta feature; to make the data ready for classification-specific tasks, we need to promote cluster identifier into target variable (a class) by reassigning the feature types in Select Columns.



The data is now ready for classification-based analysis. Here, we used a rank widget.

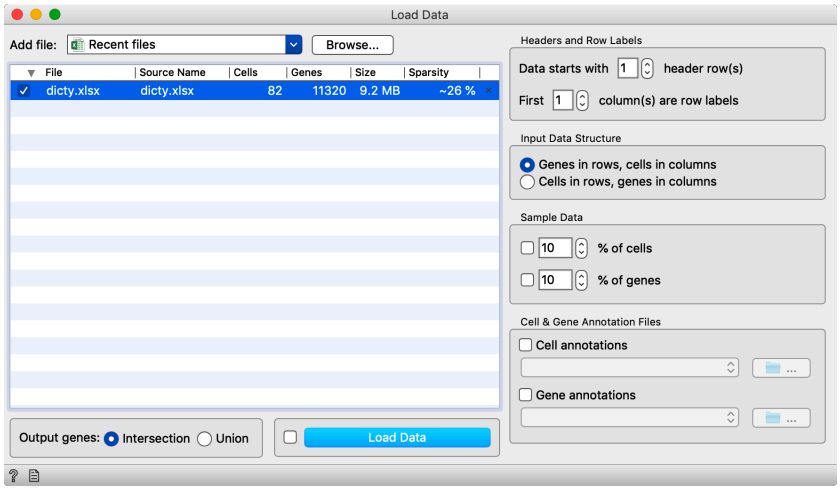


Lesson 14: Hierarchical Clustering on Single Cell Data

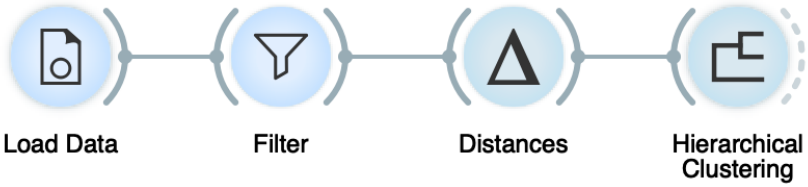
Let us apply our clustering skills on a recently published data on a social amoeba called Dictyostelium on "Cell Cycle Heterogeneity Can Generate Robust Cell Type Proportioning". Go to sciencedirect.com and search for "Nicole Gruenheit" to find the homepage of the paper. Download the first data Excel file from the supplement. The file has multiple worksheets, with the first one containing the count data, and the second one containing normalized data. We'll work with the normalized data. To avoid confusion, let's first isolate it by copying it into a new file with a single worksheet: select all data from the "normalized data" worksheet, copy, open a new file and paste. Save this file in the desktop and name it, say, dicty.xlsx.

We could use the File widget to load this data set, but will use a more specialized widget called Load Data from single-cell add-on. Add this widget to Orange canvas, open it, and then just drag the icon that represents dicty.xlsx file onto it:

Be patient! Loading Excel data files is not the fastest operation in Orange, especially for large data files like single-cell ones.



Now, filter out the genes that are expressed in less than 30 cells. Next, we will set up the pipeline for clustering. We will use the cosine distance (in Distances widget) and Ward linkage for hierarchical clustering:



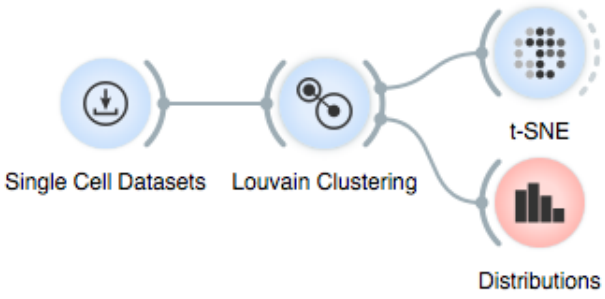
How many clusters are in this data? Compare the assignments of clusters to those that were reported in the paper (you will find the cluster assignment in one of the sheets in the downloaded Excel file). Are the clusters separable in the t-SNE visualization? How about in the visualization defined by first two principal components?

Instead of Hierarchical Clustering, check out k-means clustering Louvain clustering. Take some time to explore these two widgets. Perhaps, given time, we would say more about these two clustering methods in the class, but if you are just a reader of these lines, you could check out related videos on the Orange's YouTube channel, where k-means clustering is explained in the detail.

Lesson 15: More Clustering and Population Discovery

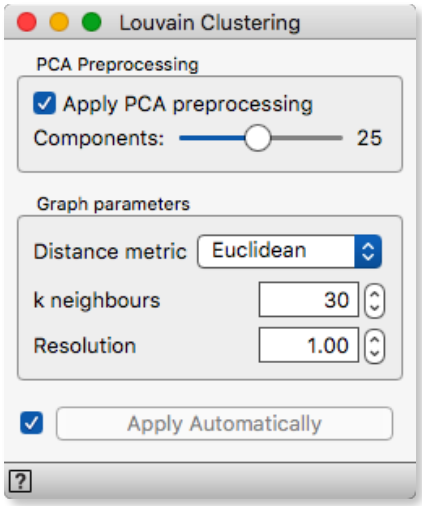
We will again use a sample from Bone marrow mononuclear cells with AML with 1000 cells that contain 1000 most variable genes.

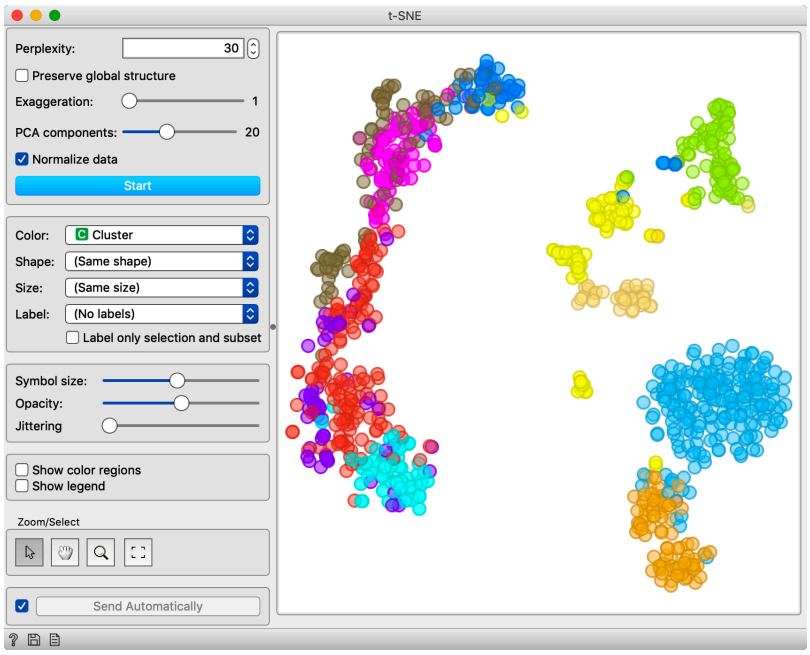
A typical task with a huge single cell dataset is to automatically determine clusters of cells. An advanced method that does this is Louvain clustering. Given the expression matrix, Louvain clustering creates a network of cells based on pairwise distances. Then, it searches for local communities — the parts of the network that are more strongly interconnected than expected by chance (think of friendships on social networks). Each local community is a cluster and genes are assigned cluster labels accordingly.



In Orange, *Louvain clustering* is in its own widget that appends a column of cluster labels to the cell data. Quite neatly, the number of clusters is determined automatically. Let us construct a workflow that displays the results of the clustering in the t-SNE plot and that examines the frequency of the cells in each of the clusters. Let us observe the t-SNE plot first.

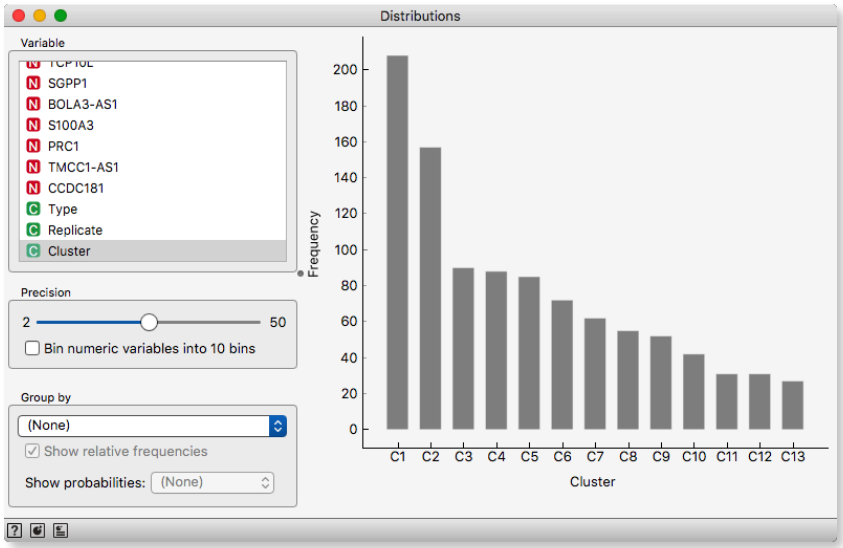
Louvain Clustering has a number of parameters. Here, we will stay with defaults, but you can experiment, change them and see the effect in t-SNE visualization.



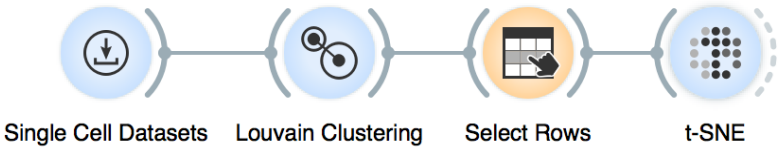


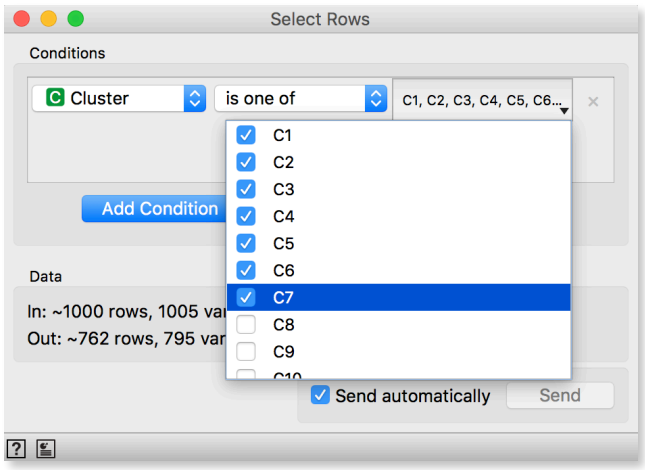
We have colored the points (cells) in t-SNE according to the cluster membership. Notice a nice separation of the clusters in the t-SNE plot. It looks like the cells are also well-separated in the original space of features (genes). A common mistake would be to compute the data projection first and then cluster the projected points. Obviously, then, the clusters would be separated perfectly and there would be no overlap.

We will now use *Distributions* widget to observe the frequency of the cells within each cluster.



We can see that the number of cells in a cluster is quite unbalanced, ranging from ~220 to a few tens of cells.





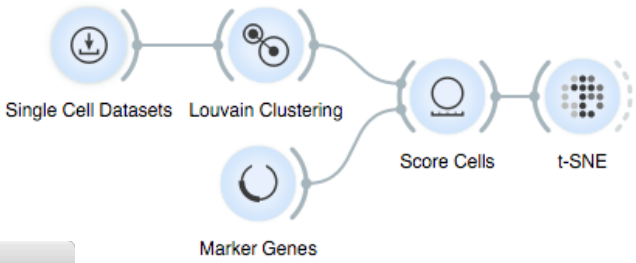
We can focus on the larger clusters (and, in a way, remove outliers) by removing the clusters C8 - C13. Hint: compare the effect of this filtering using a t-SNE plot (see screenshot on the left)!

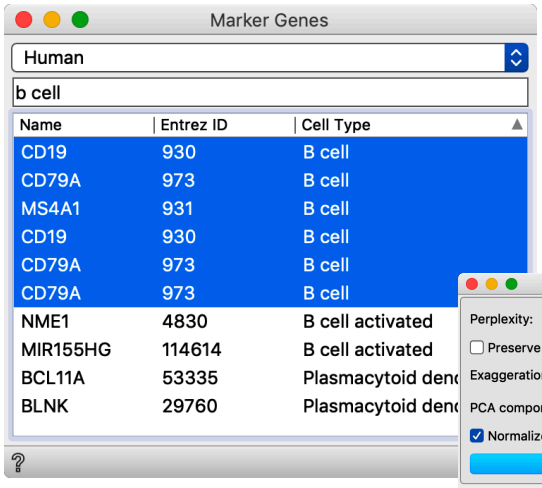
Lesson 16: Differential Gene Expression

How to tell whether clusters make biological sense? One way is to use marker genes, as before, to identify interesting clusters. And then we have to analyze what, besides the marker genes, makes the cells in the clusters special.

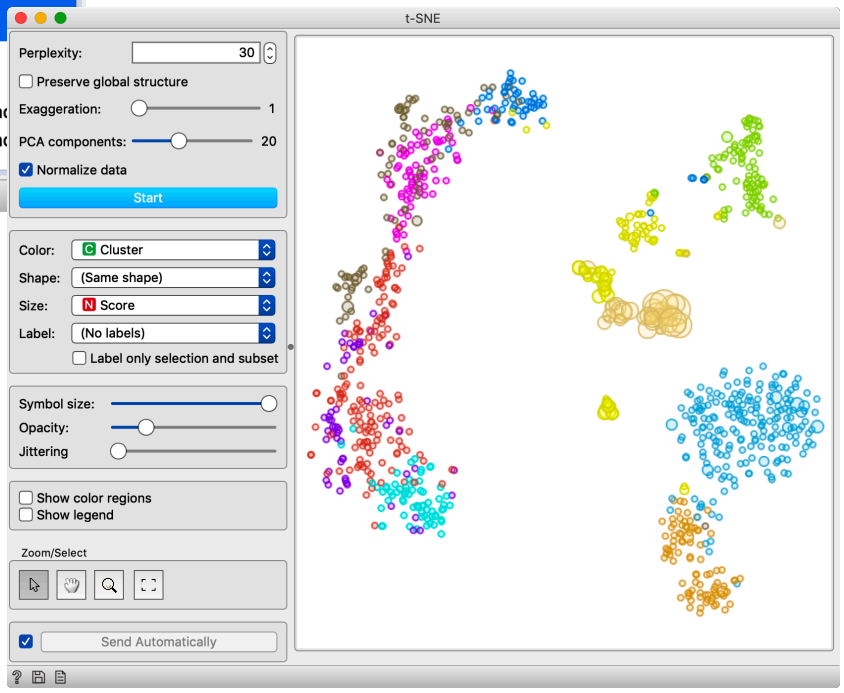
For this analysis, our staring workflow will use clustering, cell scoring and t-SNE visualization:

We could first score cells and then do the clustering, but in the current implementation the workflow on the right is faster as Louvain does not re-cluster every time we change marker genes.





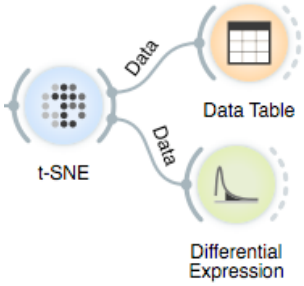
We will choose two marker genes for B cells and set the t-SNE visualization to color the points by cluster assignments and associate the size of the points to cell's score.



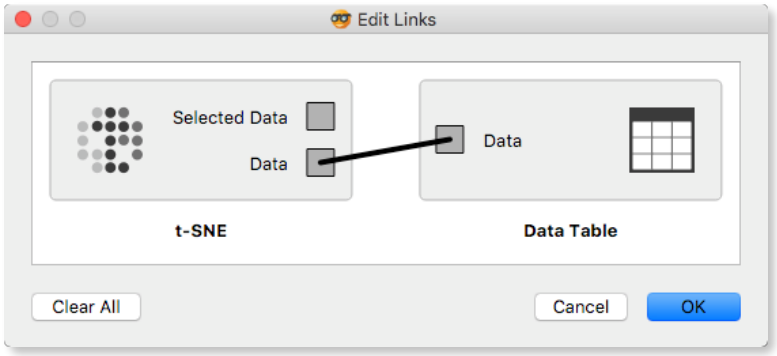
We find that B-cells are spread across several clusters (say, the green, yellow and light orange one). We can dig in further to find out which genes actually separate the clusters:

1. Use the mouse to select both B-cell clusters (use Shift+Cmd to append to clusters). On the output, in the Data channel, this selection will introduce the new column called *Selected* and mark the selected cells with *Selected* = *Yes* and all other cells with *Selected* = *No*.
2. Check the output data in the *Data Table* widget.
3. Connect the t-SNE output "Data" to the *Differential Expression* widget. Set the *Label* (in the *Target Labels* section) to *Selected* and choose *G1* and *G2* as target values.

The workflow visualization in Orange can also report on the types of inputs and outputs that get connected. Use *Preferences* from Orange menus and click on "Show channel names between widgets".



Notice that by default, Orange will connect t-SNE and Data Table through *Selected Data* channel that contains only the selected data. We need to change this, if we want to use the Data channel instead, since there are all the instances of the dataset and the data includes the Selected column. We need to rewire the connections by double-clicking on the t-SNE-Data Table channel:



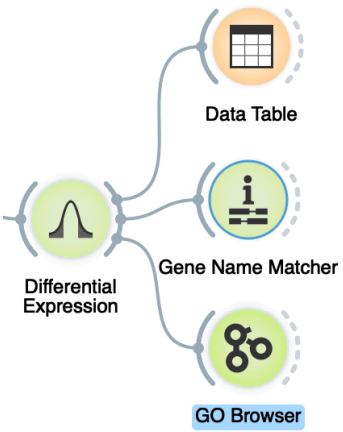
We should do the same for the connection between t-SNE-Differential Expression widgets.

Since our genes are in the columns, *Differential Expression* would output the data with columns that include only the selected genes. Check this out in the *Data Table*.

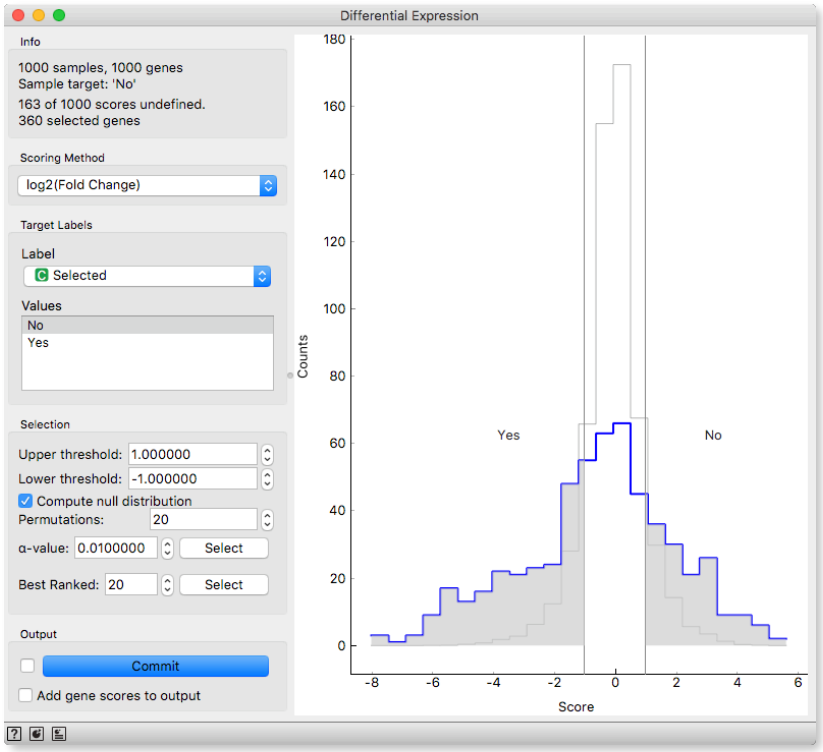
We have set the scoring method to *log2(Fold Change)* and set the *Label* to *Selected*. *Differential Expression* also compares the

Spoiler alert: see the final t-SNE projections and the complete workflow on the next page.

The distribution marked with grey line shows the *null distribution*, the distribution of genes scores expected by pure chance.



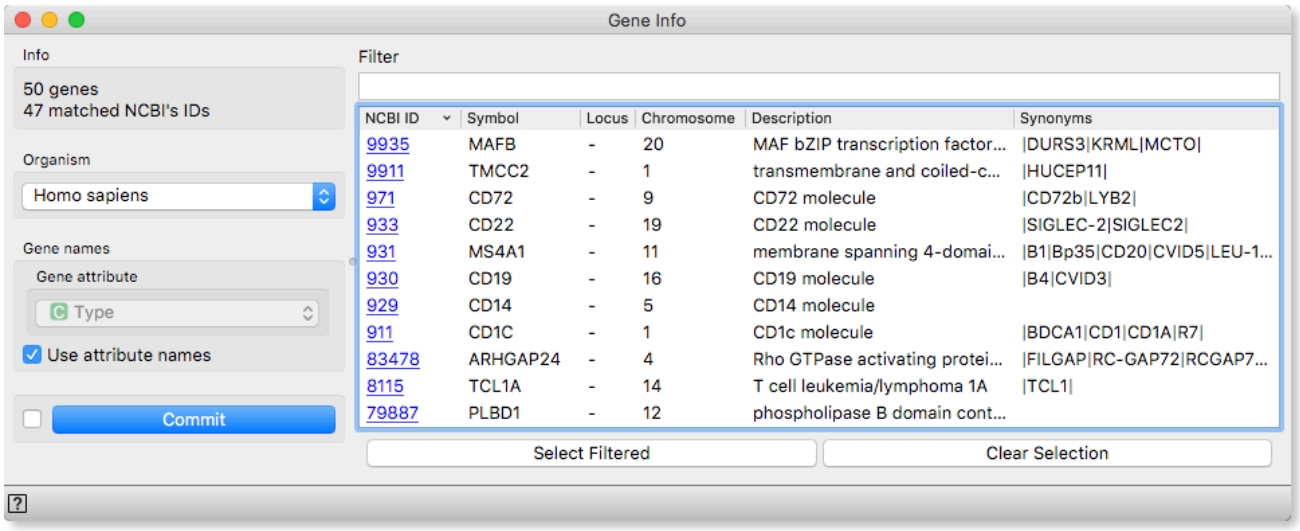
Differential expression shows the distribution of the differences of gene expression in selected and all other cells:



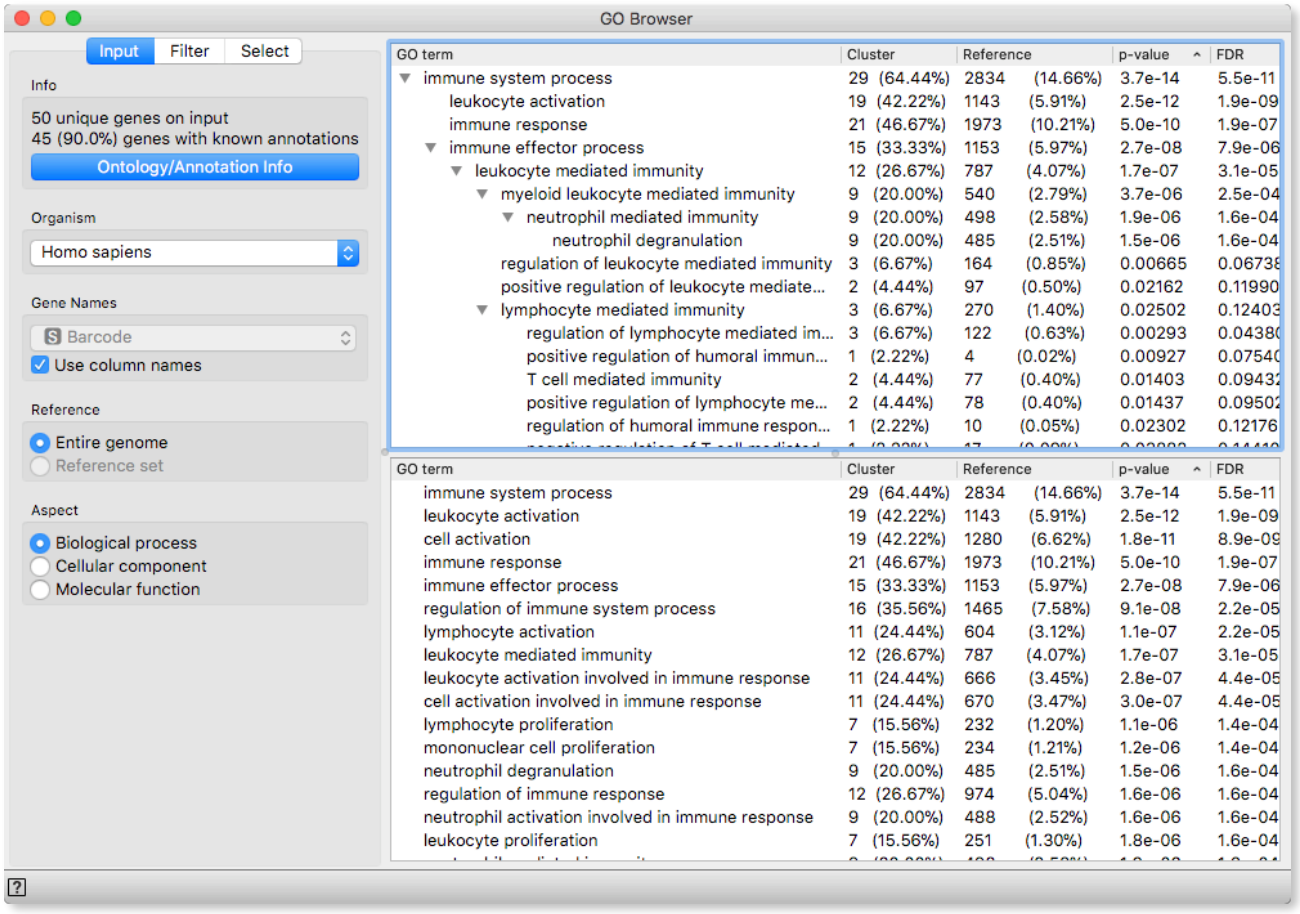
Differential Expression widget outputs the data with genes that are in extremes of the distribution. That is those, for which the difference in selected and non-selected cells is the largest. Genes that are most differentially expressed, lie on the left and on the right side of the two vertical splitters and their score value belongs to the shaded part of the distribution. Move the two vertical splitters such that there are only 50 selected genes.

So, where are the genes that are selected in the *Differential Expression* widget? In the output of the widget. We can observe the new dataset and analyze the set of selected genes with widgets that we connected to *Differential Expression*. Observe the data in the *Data Table*, a list of selected genes in *Gene Info* and the results of analysis of *Gene Ontology* term enrichment in *GO Browser*.

Gene Info displays the list of selected genes. Notice that not all gene names from the selection in Differential Expression were matched to the NCBI's database:

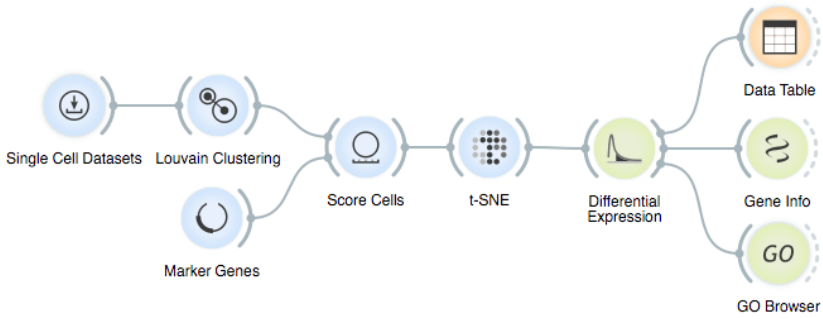


To validate the biological meaning of differentially expressed genes, we use one final widget: *GO (Gene Ontology) Browser*. Here, we compute the enrichment of biological terms that are pertinent to our subset of selected genes. For B-cells, we expected to find genes related to the immune system. And so we did.



Make sure you have selected *Use column names* in the GO Browser to indicate where to look for gene names. Also, make sure that the right organism has been selected.

Our final workflow is as shown below. Try choosing other markers, select other clusters of cells, and observe the changes in the content of the widgets in our workflow.

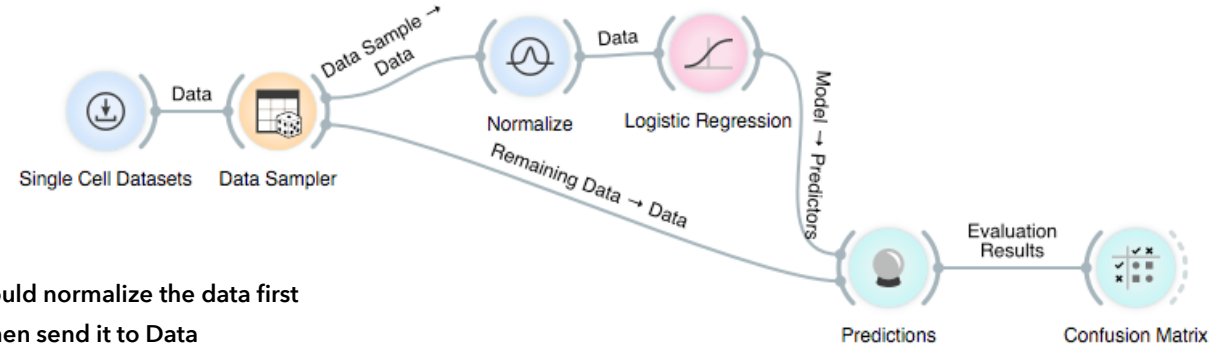


Lesson 17: Predictive Modeling

Once cell types or other cell classification are determined, automatic classification models can be used to predict the type of new, unlabelled cells. In this lesson, we will build a simple linear classification model to predict cell cycle stage and learn about the techniques to score the accuracy of our models.

Load the familiar Cell cycle in mESC (Fluidigm) dataset with 182 cells and a reduced set of pre-selected 564 genes. You can experiment with a dataset with a full collection of genes later.

To predict, we need two dataset: one on which to build a model and a separate dataset on which to make predictions. We can simulate this process by splitting our cell cycle dataset to two: the *training* and *test* set — the former will be used for model inference and the latter for evaluation. Evaluation will compare the predicted cell cycle stage to the "true" one in the test set.

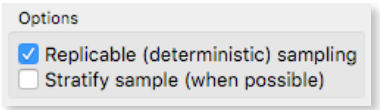


We could normalize the data first and then send it to Data Sampler. But that would be conceptually wrong. Why?

We will use *Data Sampler* widget to split the data such that 70% of the data will go into the training set and the remaining 30% of the data to the test set. Data Sampler uses two output channels, and we have to make sure that the right data channel is used. Here is our workflow with annotated communications channels:

The modeling branch of the workflow first normalizes the data, and sends the normalized data to the *Logistic Regression*, a widget that builds the predictive model. Our data includes a special column called *CycleStageName* and it is this variable that will be modeled from variables that profile the cell, that is, from gene expressions.

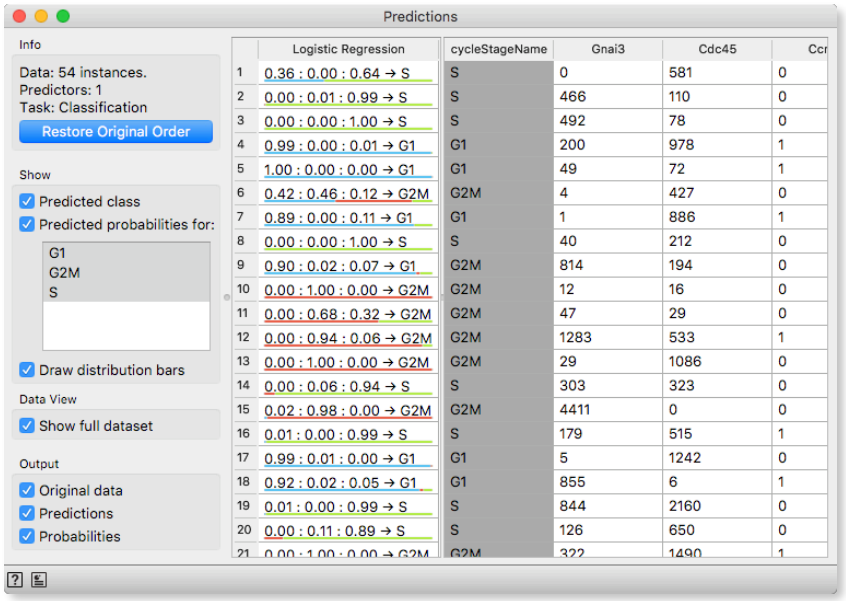
We can force Data Sampler to always return the same sampling under the same sampling parameters by choosing *Replicable (deterministic) sampling*.



The prediction branch of our workflow takes the remaining data and uses the developed model to predict the cell cycle stage. The predictions are made in the *Predictions* widget. Logistic regression predicts probabilities for all the classes in the input data and we

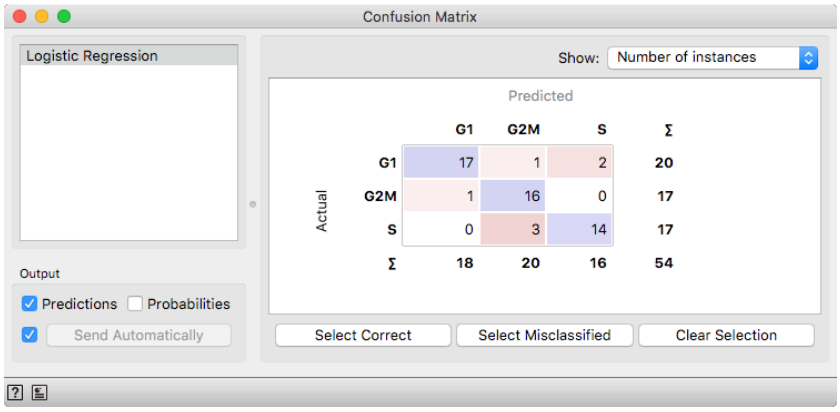
To compare predictions with the true class, compare Logistic Regression column to the column CycleStageName. The widget *Predictions* also outputs a data table with prediction results that store probabilities and predictions and these may be analyzed in other Orange's widgets.

can observe these in the *Predictions*:



Notice that the predictions (Logistic Regression column) are mostly correct. There is one error in row 9 where G1 was predicted instead of the true class G2M and where the probability assigned to the true class by logistic regression was only 0.02.

It would be hard to find all correct and false predictions from the table provided in *Predictions* widget. A better way to summarize the results of prediction is with *Confusion Matrix*:



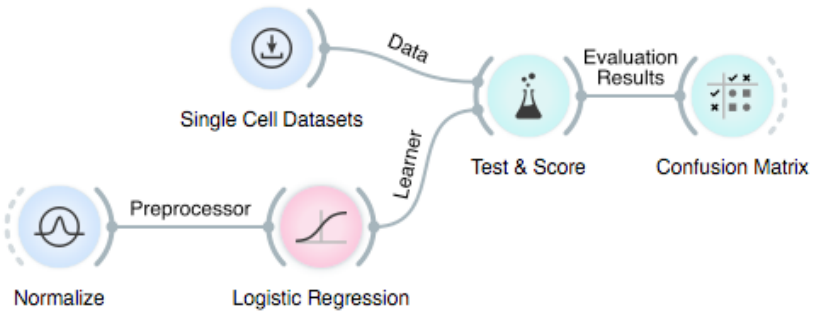
You may try to change the dataset now to the one that includes the full complement of genes. What do you notice?

Out of 54 cells, logistic regression mis-predicted the class for seven cells.

Lesson 18: Cross-Validation

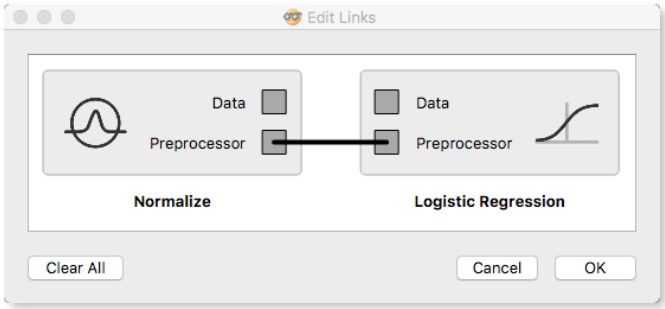
The results of the classification depend on the training set and the accuracy of the test set, of course, depends on the composition of the test data. For speed, change the dataset to the cycle cell data with a subset of genes. Then, open the *Data Sampler* and the *Confusion Matrix* widget and press Sample button in the Data Sampler. See how the classification results change with every fresh instance of the data sampling?

Classification results depend on sampling. When reporting how good is our data and modeling approach, we should sample many times and report on average accuracy. There is a *Test & Score* widget that does this for us and a particular sampling technique called cross-validation that became a standard within machine learning. In cross-validation, the data is split to folds and in each model-and-testing iteration, each fold is used for a test dataset exactly once. The cross-validation workflow is:



Logistic Regression is only one of the many learners in Orange. Another very popular one is *Random Forest*. Try to include it in our workflow!

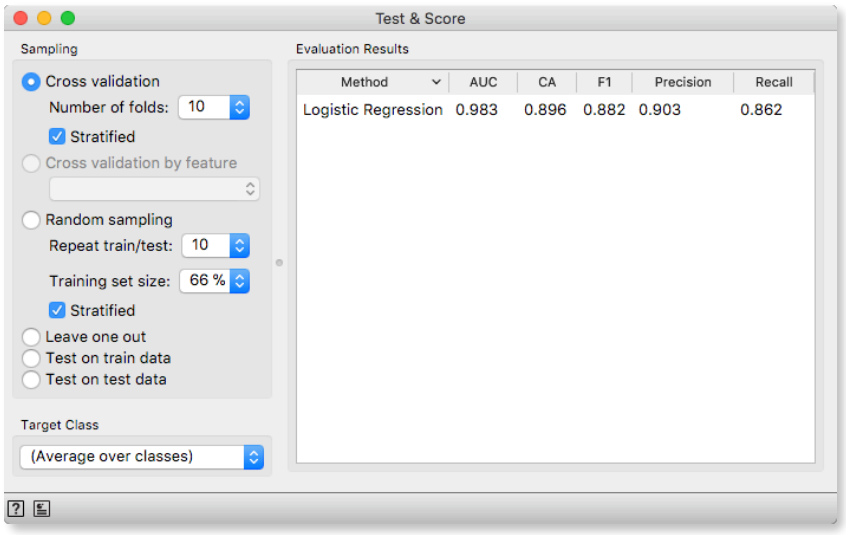
Make sure that Normalize sends a preprocessor to Logistic Regression. Orange will ask what signals these two widgets should exchange upon connecting them. You can always change the type of connection by double-clicking on it.



It is the *Test & Score* widget that does data sampling, model construction, and testing. Hence, *Test & Score* needs a method that is used for modeling and this is provided by *Logistic Regression* as a *Learner*. Notice also that we should execute normalization only on the training data, as considering the test data at that stage would be like seeing the future and hence cheating. Normalization is hence a part of the learning and a component that is executed prior to logistic regression. In Orange, this is solved by providing normalization as a preprocessor that logistic regression — or any other modeling technique — includes in the model development procedure.

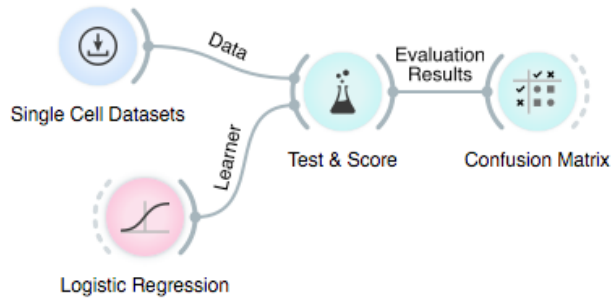
And the results? Open the *Test & Score* widget. It reports on several classification accuracy statistics. The column CA stands for classification accuracy: our predictions were right about 90% of the time. Another often used score is the area under the ROC curve. AUC considers class probabilities, is a discriminative measure and is more reliable than classification accuracy as the score does not depend on class distribution in the dataset. Our models, on average, scored very well:

Test & Score evaluates the predictive performance by counting how many of the test labels our classifier got correct. In a multi-class setting, this is done on a one-versus-all basis.



Each of the learning algorithms has parameters. Double-click on Logistic Regression or Random Forest (if you have used this learner) to change them.

For a final test, remove the normalization. How do accuracy results depend on this component of data preprocessing?

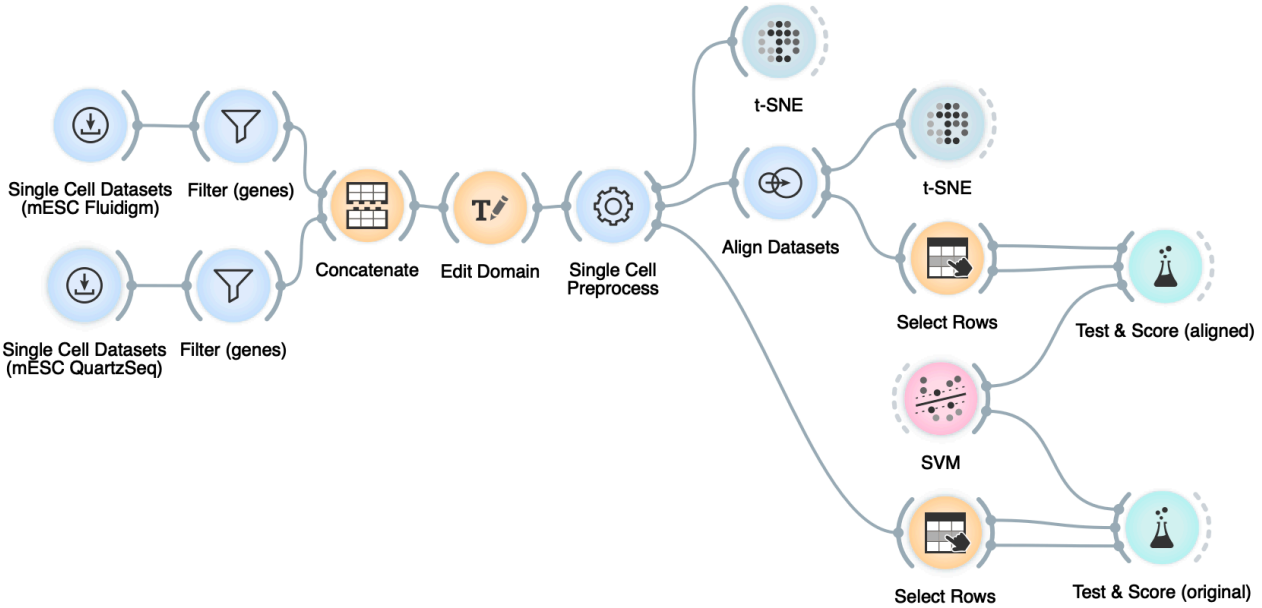


Lesson 19: Data Set Alignment & Cross-Dataset Classification

In the previous lecture we classified cells from the same batch. In other words, we use the data from the same experiment to train the classifier and to test it. One of the main challenges in the single cell area is to use one data set to build the classifier, and then to classify the cells from another experiment. The goal would be that anyone can use expression profiling of the cells, and then some previously developed classifier to find which types of the cells she is dealing with.

There are many problems related to cross-data set classification, but the principal one is that the differences between two experiments are often larger than the differences between the cell types within each of experiments. We need to align the data sets to remove cross-experiment differences.

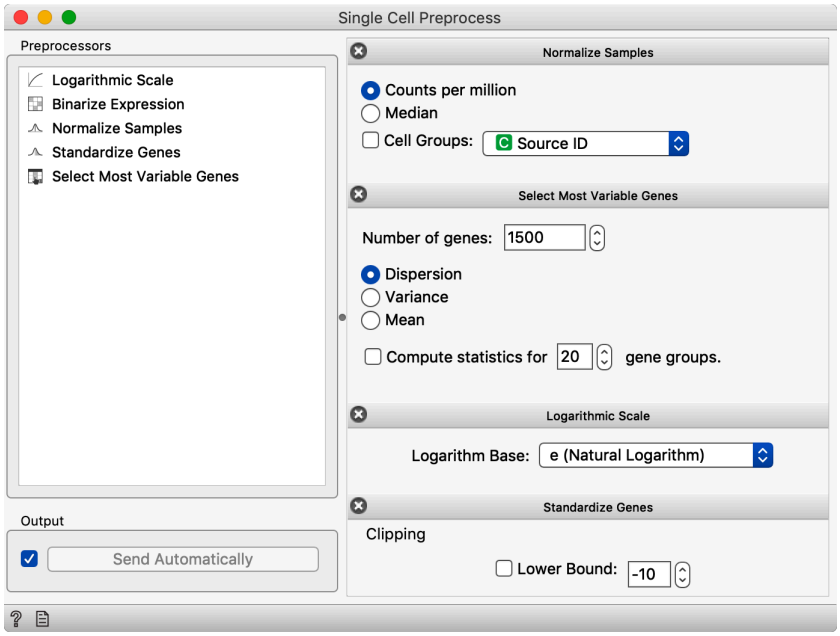
We will consider two different data sets that each include mouse embryonic stem cells that are stage-labeled. The idea is to train stage model in one data set, and test the stage prediction on the other data set. The workflow for this last lesson is the most complex so far:



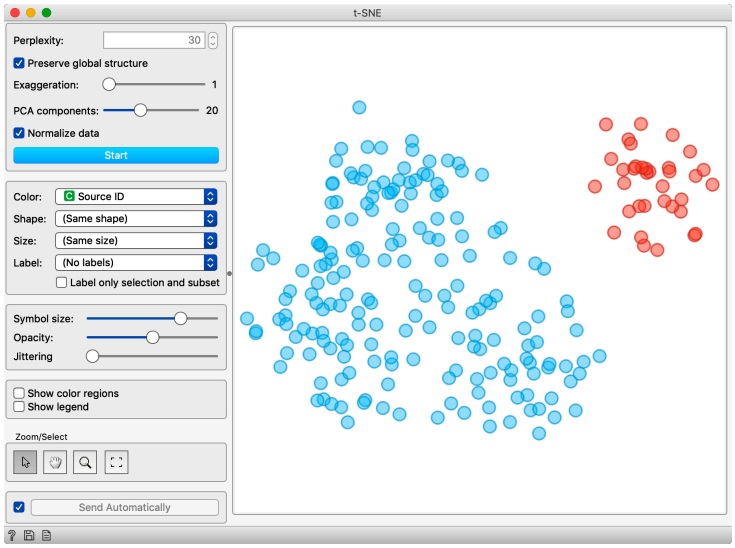
We consider two mESC data sets, one obtained by Fluidigm, the other by QuartzSeq. The first data is the one we got familiar with already, while the QuartzSeq data set includes 35 cells, again classified into one of the G1, G2M, or S stages.

The workflow, in short, first reads both data sets. Then we filter Fluidigm data (genes, detection count in between 10 and 182 cells) and QuartzSeq data (genes, detection count in between 3 and 30 cells). We concatenate both data sets and include only about 7300 intersecting genes.

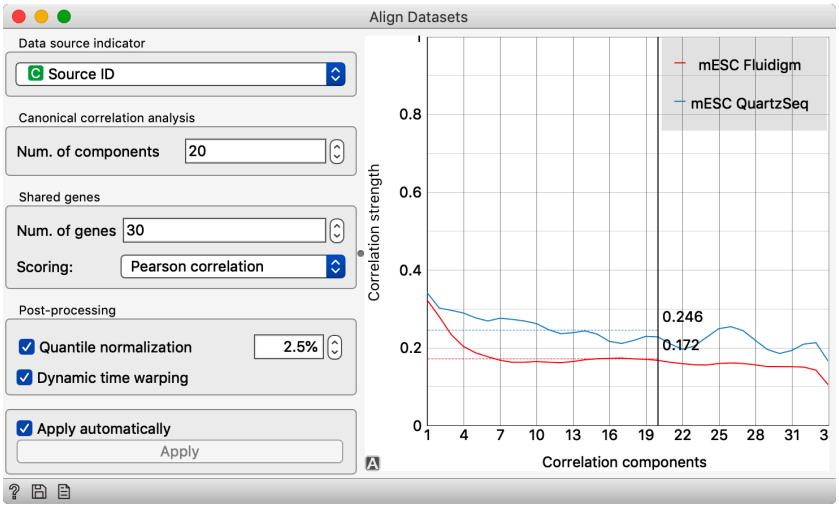
The preprocessing uses the following standard pipeline:

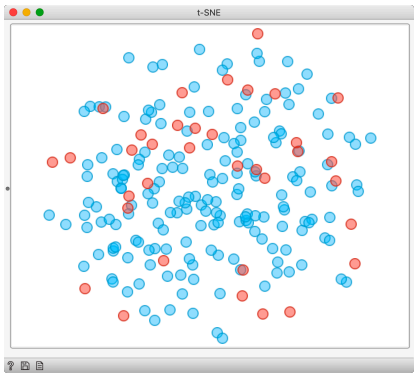


The t-SNE visualization after all this steps shows that the two data sets are quite different to each other.

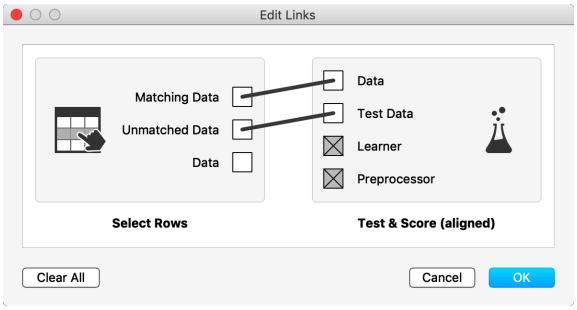


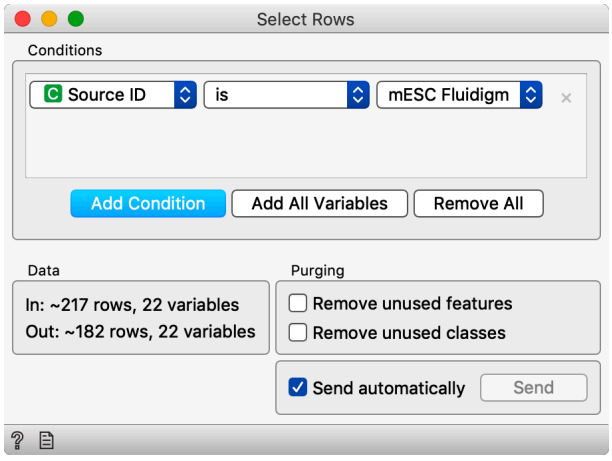
We need to align the two data sets. We will use alignment by canonical correlation analysis, as originally implemented in Seurat R library:



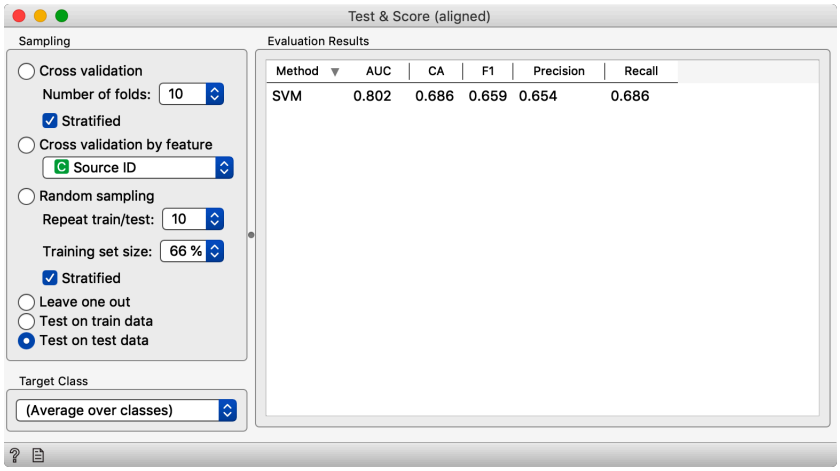


The t-SNE after alignment shows the mixing of cells from the two experiments. Now, the only question is if the alignment has helped us in getting a better classification accuracy. We need to split the data again, using Fluidigm for training and QuartzSeq for testing. We do this through selecting only Fluidigm cells in Select Rows widget, and then feeding the selection and all other data into Test & Score. Double click on the link between Select Rows and Test & Scores to set the communication between these two widgets accordingly.





The AUC of predicting QuartzSeq cells using the SVM model developed on the Fluidigm data is around 0.8:



This is actually quite high, considering how different are these two data sets. We get a much lower accuracy on the non-aligned data, with AUC around 0.7.

For the End

Our single-cell Orange Data Mining course ends here. We covered quite a lot of topics and hope we have taught you some crucial algorithms that should be on the stack of every single-cell data scientists. The course, though, was only an introduction. There are many of more advanced topics that we left out, and these include imputation, removal of batch effects, other data visualization tools, and cross-data cell classification. Our goal was also to expose you to various aspects of data science, and to show you how to enjoy it through interactive visualizations and construction of workflows.