DataMole User Manual

Alessandro Zangari

October 3, 2020

Contents

List of Figures

1 Overview

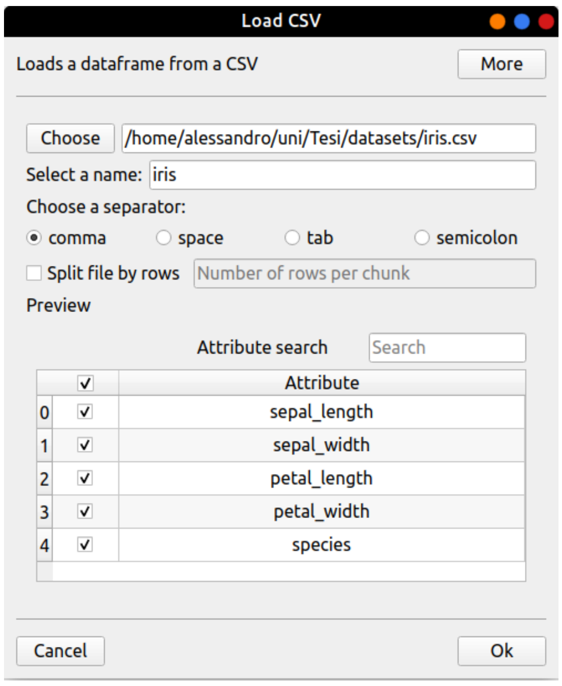
This guide describes how to use DataMole for data analysis. The tool allows to load tabular datasets from disk and apply transformations to their columns. This manual does not include a list of available transformation, nor does it explain how to use them, since this information can be accessed directly while using DataMole through a specific help widget.
A section describing how to install and launch DataMole was omitted, since this information is kept updated in the `readme` file of the GitHub repository.

1.1 Data manipulation features

One of the DataMole core features is the ability to apply *transformations* to the dataset. The following transformations are available:

- *Type conversions*: as explained in §**??** types are automatically inferred when a file is parsed, but the program allows to explicitly set types and convert between them if needed;
- *Dataset join*: it is possible to join pairs of datasets on specific columns or alternatively on index columns. Inner, outer, left and right SQL-like join is supported;
- *Missing values management*: NaN values can be imputed with different strategies, for example by replacing them with the column average value, with a specific value or with the last valid value. Additionally rows or columns with a high number of missing values can be removed;
- *Indexing*: sometimes it is useful to set one or more attributes as indices of the dataset, for instance, before joining two datasets;
- *Scaling*: attributes can be scaled to a specified range with *min-max scaling* or standardised with respect to their mean and standard deviation;
- *Discretization*: continuous features (i.e. of type numeric or datetime) can be discretized into a variable number of bins with different strategies, including equal-sized bins, equal-frequency bins and manual range specification;
- *One-hot encoding*: nominal categorical features and string attributes can be one-hot encoded;
- *Cleaning operations*: attributes values can be replaced with different values, columns can be renamed, duplicated and dropped;
- *Time series extraction*: one of the requirements for this project was the ability to visualise the temporal information contained in longitudinal datasets. To be interpreted as a time series, this information must first be extracted with an operation designed for this purpose.

Most of these transformations are quite standard in the data science domain. On the other hand, the *time series extraction* feature was specifically designed to work with longitudinal datasets and its purpose is described with an example in §2.7.



/home/alessandro/uni/Tesi/datasets/iris.csv

Select a name: iris

Choose a separator:

[✓ comma] space tab semicolon

Split file by rows: Number of rows per chunk

Preview

Attribute search: Search

	✓	Attribute
0	✓	sepal_length
1	✓	sepal_width
2	✓	petal_length
3	✓	petal_width
4	✓	species

Figure 1: The widget used to load a `CSV` file

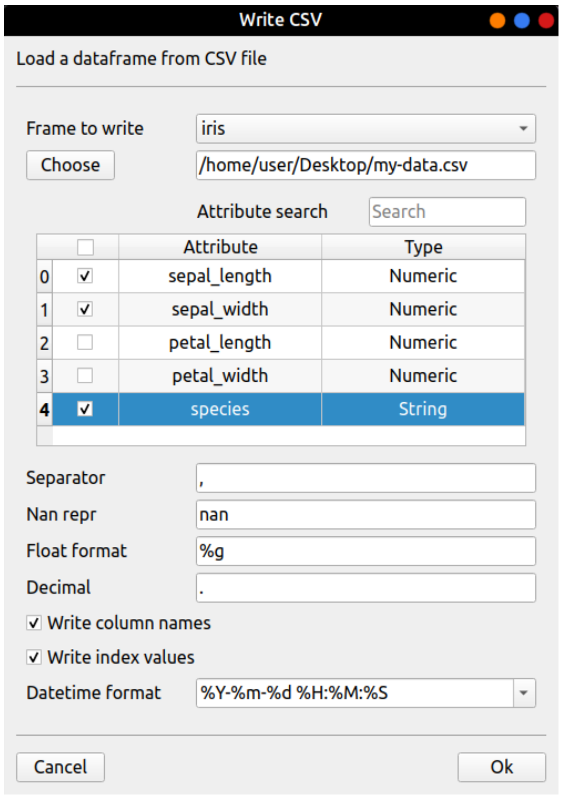
2 Using DataMole

2.1 Importing a dataset

DataMole can import tabular datasets from `CSV` and `pickle` files that contain serialised Pandas dataframes. Notice that `pickle` files may contain any Python object, but only Pandas dataframes can be loaded in DataMole.
By clicking `File > Import > "From csv"`, the editor shown in Fig. 1 appears. It allows to choose the file separator and to select which columns to load. Columns can be selected in the provided table, that, depending on the size of the dataset, may require some time to be shown. Big datasets can be loaded in multiple dataframes, by selecting *"Split file by rows"* and specifying the maximum number of rows per dataframe. Clicking the *"More"* button opens a side panel with additional information on the operation.
Similarly, the widget to load a `pickle` dataframe can be opened clicking `File > Import > "From pickle"`.
Every imported dataset will be visible in the *workbench*, the widget that lists all loaded dataframes, visible in ??.

2.2 Exporting a dataset

Loaded dataframes can be exported in `CSV` or `pickle` files. The latter option is useful to continue working on the dataset outside of DataMole, because every Python script can be used to load these files.
Fig. 2 shows the widget for `CSV` export. DataMole can load multiple dataframes, thus it is required to select which one to export using a combo box. Clicking on *"Choose"*



Write CSV

Load a dataframe from CSV file

Frame to write: iris

Choose: /home/user/Desktop/my-data.csv

Attribute search: Search

		Attribute	Type
0	✓	sepal_length	Numeric
1	✓	sepal_width	Numeric
2		petal_length	Numeric
3		petal_width	Numeric
4	✓	species	String

Separator: ,

Nan repr: nan

Float format: %g

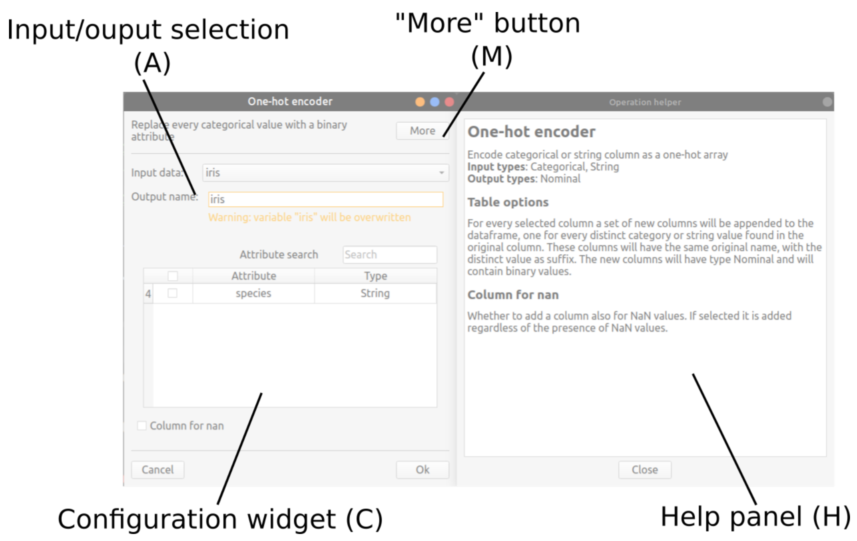
Decimal: .

✓ Write column names

✓ Write index values

Datetime format: %Y-%m-%d %H:%M:%S

Figure 2: The widget used to export a dataframe in `CSV` file



Replace every categorical value with a binary attribute

iris

iris

Warning: variable "iris" will be overwritten

	Attribute	Type
4	species	String

One-hot encoder

Encode categorical or string column as a one-hot array
Input types: Categorical, String
Output types: Nominal

Table options

For every selected column a set of new columns will be appended to the dataframe, one for every distinct category or string value found in the original column. These columns will have the same original name, with the distinct value as suffix. The new columns will have type Nominal and will contain binary values.

Column for nan

Whether to add a column also for NaN values. If selected it is added regardless of the presence of NaN values.

Figure 3: The *editor widget* used to configure the one-hot encoder and its help window

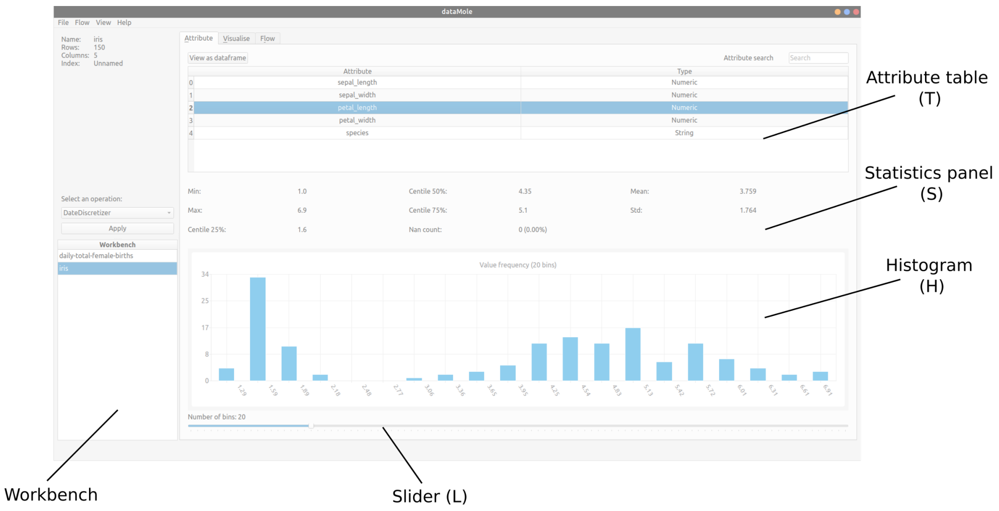
opens a dialog window for selecting the save path location. The attributes to be included can be selected in the table, and many common options can be set.

2.2.1 Applying operations

When an operation is selected in (A) and the *"Apply"* button is pressed, the operation *editor widget* is opened, like the one shown in Fig. 3, used to one-hot encode columns. Every editor has an header with a brief description of the operation and a *"More"* button on the right (M). By clicking on it, an help window with information on the operation and on the required parameters will be opened (H).
Below the header, a combo box allows to select the dataset to transform and a text box can be used to insert the name of the output dataset (A). After the options are confirmed with the *"Ok"* button the operation will be applied and its result will be written in a *workbench* variable with the specified output name. The output name defaults to the input name, thus the transformed dataset will replace the original one if the name is not edited.
Depending on the operation many options may need to be configured in (C). In the example the operation only requires to select the columns to encode and whether or not NaN values should be considered.

2.3 The Attribute panel

This panel is displayed on the left side of Fig. 4, positioned inside the tabbed widget. It provides some basic features to get an overview of the content of a dataset.
The *attribute table* (T) displays the column names and their types: every row represents a column (i.e. an attribute) of the dataset. Column names can be changed by double clicking the cell and typing in the new name. All column names should be unique, so duplicated names will be rejected.
Above the table a search bar allows to search through attributes. Search by regular



Name: iris
Rows: 150
Columns: 5
Index: Unnamed

	Attribute	Type
0	sepal_length	Numeric
1	sepal_width	Numeric
2	petal_length	Numeric
3	petal_width	Numeric
4	species	String

Min:	1.0	Centile 50%:	4.35	Mean:	3.759
Max:	6.9	Centile 75%:	5.1	Std:	1.764
Centile 25%:	1.6	Nan count:	0 (0.00%)		

Figure 4: The main window set on the *Attribute panel*

expression is also supported using Perl-style syntax. The search is case-insensitive.
The *statistics panel* (S) and the *histogram* (H) show information about the currently selected column, that can be changed by clicking on a row of table (T). The histogram shows the number of occurrences of every distinct values for *string* and *categorical* attributes, while *numeric* or *datetime* are first discretized in a predefined number of equal-sized intervals. This number can be changed by moving the slider below the chart (L).

2.4 The View panel

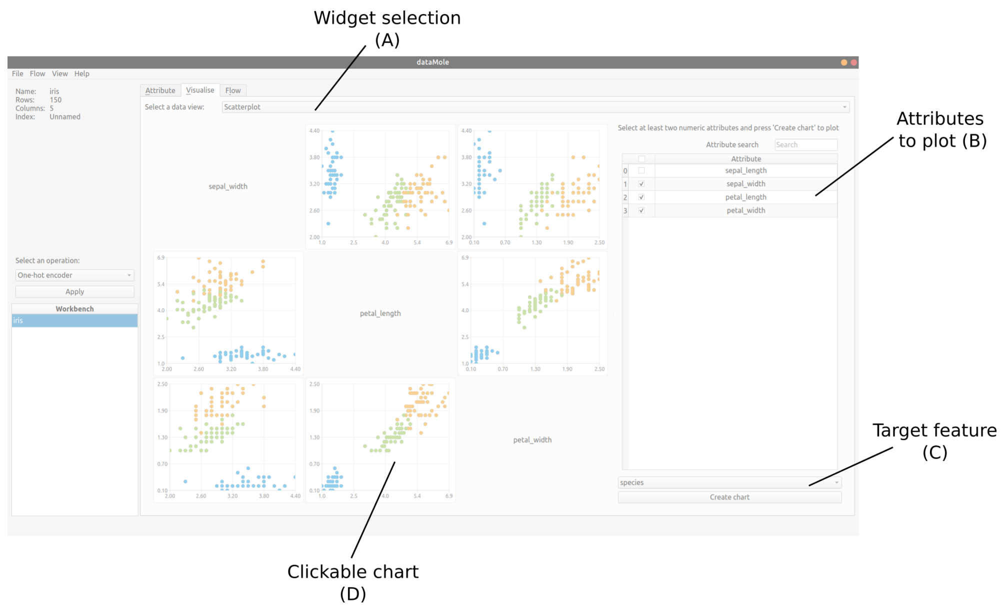
This tab groups DataMole visualisation features. Currently it supports the creation of a scatterplot matrix, to inspect feature correlation, and a line chart to plot time series.

2.4.1 Scatterplot matrix

A scatterplot matrix with 3 attributes is shown in Fig. 5. A dataset must be selected in the *workbench* and the attributes to plot can be chosen in widget (B). Scatterplot dots are usually coloured differently depending on the values of a target attribute, that can be chosen using the combo box below the table (C).
Double clicking a scatterplot opens it in a new window, like in Fig. 6. Here dots can be hovered to inspect their values and the chart can be zoomed and stretched as required. The content of the window can additionally be saved as an image in different formats (`PNG`, `JPEG`, `BMP`, `XMP`).
The combo box in (A) allows switching between the two chart types.

2.4.2 Time series plot

Time series can be represented in a line chart like in Fig. 7 using this widget. First, the dataset must be selected in the *workbench*. Then the attribute that contain the time axis labels must be selected in (A): it must be either an attribute with type



Name: iris
Rows: 150
Columns: 5
Index: Unnamed

Select at least two numeric attributes and press 'Create chart' to plot

Figure 5: A scatterplot matrix with 3 attributes

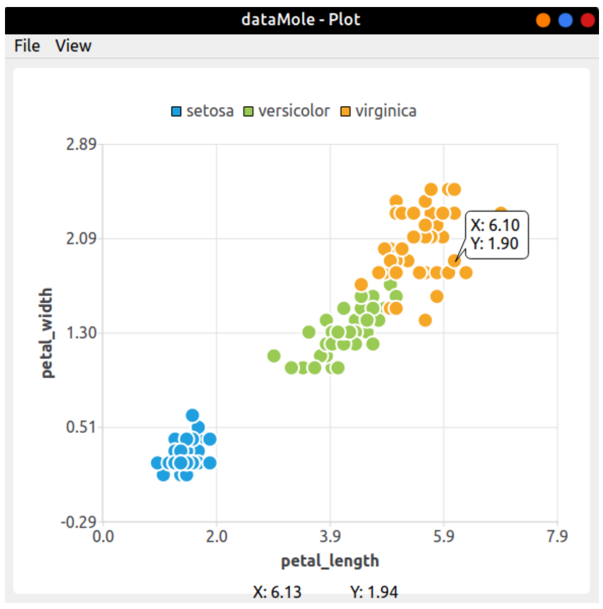


Figure 6: A scatterplot displayed in a new window

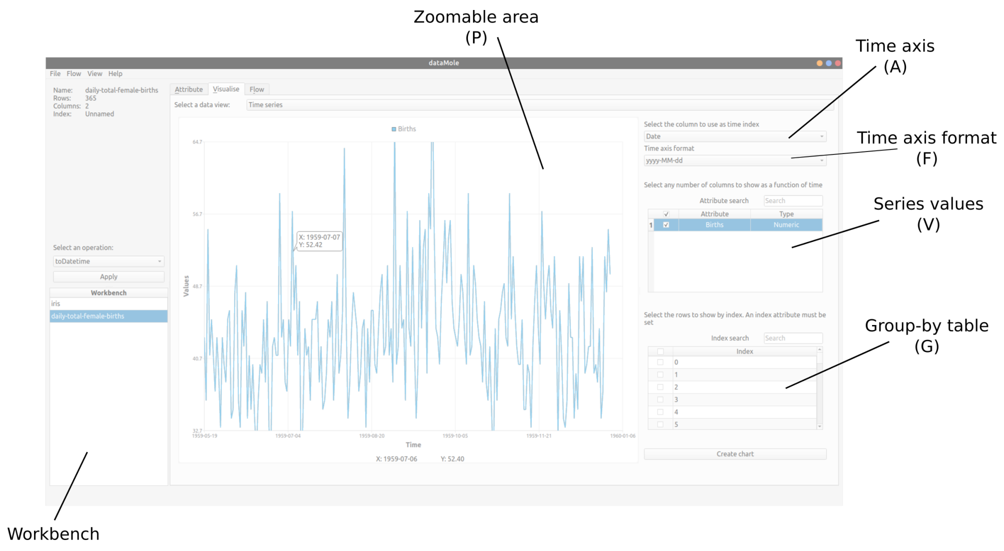


Figure 7: A time series displayed with a line chart

datetime or with *ordinal* type, since a order must be defined between its values. If a *datetime* attribute is selected the label format for visualisation in the horizontal axis can be changed in (F).
The time dependent attributes can then be selected in table (V). The chart area (P) is interactive and can be zoomed and moved around. The initial state of the chart can be restored by using the `Ctrl+R` shortcut.
If an index is set in the dataframe, table (G) can be used to select a subset of indices to plotted. When the chart is created, data are grouped by index, and every group is considered a different time series to plot. This feature was included in order to plot time series which have been extracted by longitudinal datasets. The documentation of the *Time series extraction* operation provides more details about this.

2.5 The Flow panel

The last available panel provides an alternative approach to dataset transformation: the same operations that could be applied singularly from both the *Attribute* and *View panel* can be chained together to form a pipeline where the output of a node becomes the input of its successors. Fig. 8 displays a pipeline composed of 5 steps.
Steps can be added to the pipeline canvas (P) by dragging them from the list of operations on the left side of the window (A). Every step is represented graphically with a node that has some *knobs* (K) on the left an right side. Nodes can be connected by clicking a *knob* on the source node and dragging the interactive edge up to the target node, where it must be dropped. Most operations require some parameters before the pipeline can be executed. In order to do this every operation has a *widget editor* that is used to configure them. For instance the widget shown in Fig. 9 is the editor widget for the min-max scaler operation. These editors can be opened by double clicking the operation nodes in the pipeline, and when options are set they can be confirmed with the *"Ok"* button. At this point the operation is configured an its *options indicator* (C) turns green.
The pipeline canvas can be zoomed and moved around using the mouse wheel. Pressing

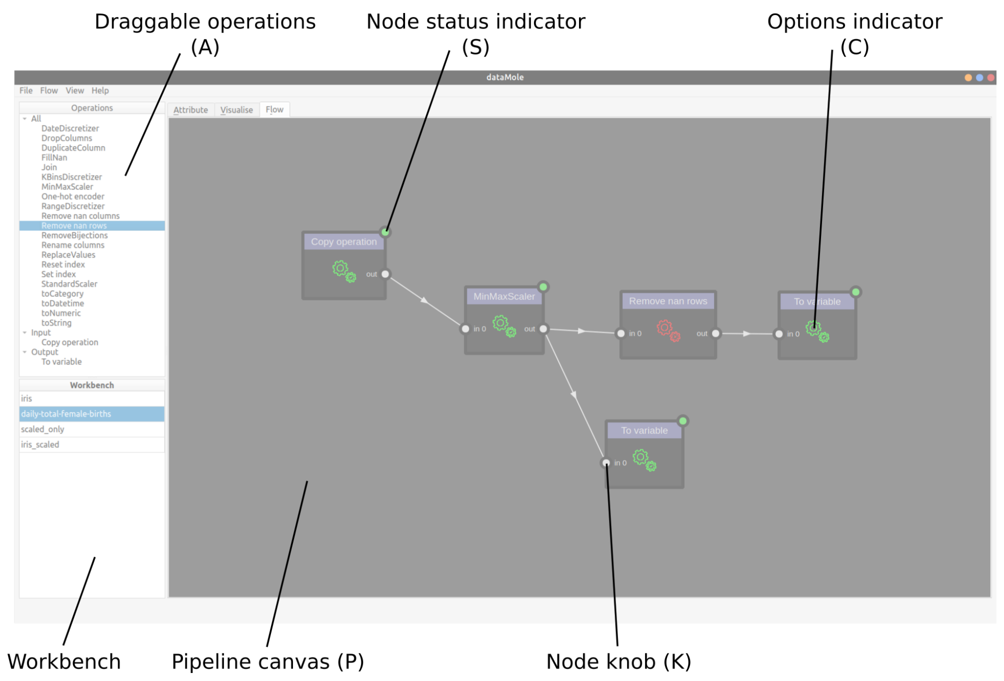


Figure 8: A simple pipeline defined in the *Flow panel*

the `F` key while the canvas is focused fits the view to its content.
When the pipeline has been configured it can be executed by clicking `Flow > Execute`. Once started, a *status indicator* appears above every node (S). It is grey for nodes that must still be executed, green for completed nodes, yellow for running nodes and red for nodes that failed with an error. This status can be reset by clicking `Flow > "Reset status"`.
Pipelines can also be imported and exported using the respective entries in the menu bar: `Flow > "Load"` and `Flow > "Save"`.

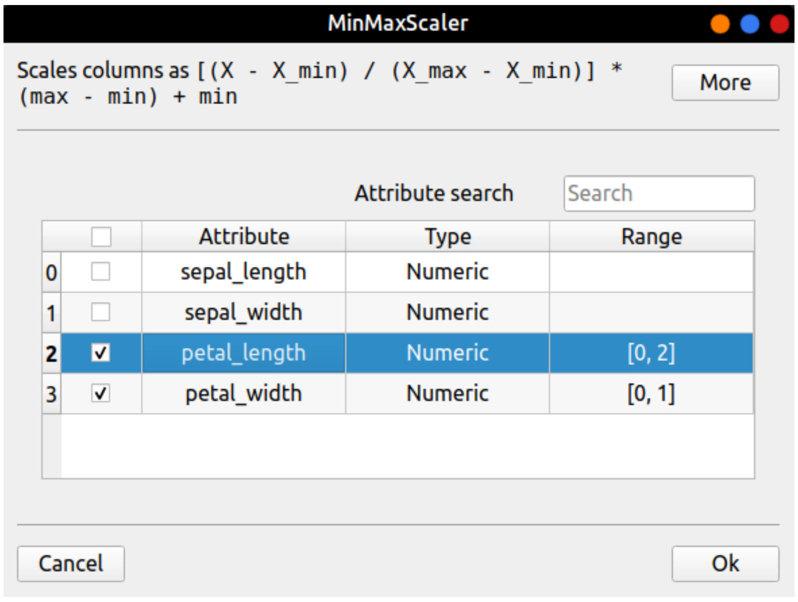
2.6 Other features

2.6.1 Dataframe visualisation

Menu entry `View > "Compare dataframes"` opens a window where two dataframes can be selected and compared side by side, like in Fig. 10. It is also possible to visualise single dataframes in a similar table, by clicking button "View as dataframe" in the *Attribute panel*.
These views are read-only, thus datasets can not be edited from within DataMole.

2.6.2 Logging facilities

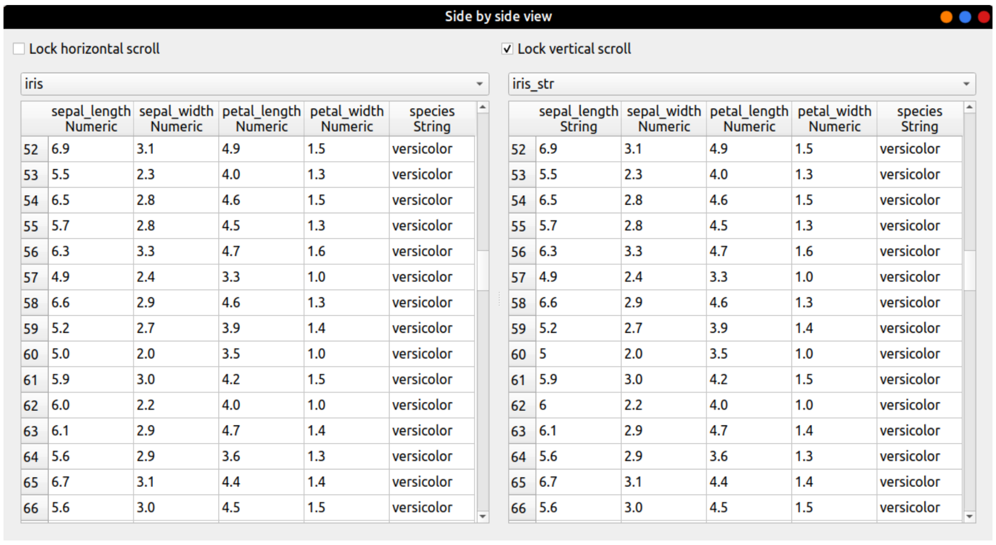
Whenever a pipeline is executed from the *Flow panel* DataMole writes an execution report inside the `logs/graph` folder. Every log file is named with the timestamp of when it was created and includes information on the configuration of every operation (like user supplied options) and eventual parameters computed during execution. Also operations applied singularly from the *Attribute panel* are logged in the same way



Scales columns as `[(X - X_min) / (X_max - X_min)] * (max - min) + min`

	☐	Attribute	Type	Range
0	☐	sepal_length	Numeric	
1	☐	sepal_width	Numeric	
2	✓	petal_length	Numeric	[0, 2]
3	✓	petal_width	Numeric	[0, 1]

Figure 9: The *editor widget* for the operation used to scale columns



☐ Lock horizontal scroll

iris

	sepal_length Numeric	sepal_width Numeric	petal_length Numeric	petal_width Numeric	species String
52	6.9	3.1	4.9	1.5	versicolor
53	5.5	2.3	4.0	1.3	versicolor
54	6.5	2.8	4.6	1.5	versicolor
55	5.7	2.8	4.5	1.3	versicolor
56	6.3	3.3	4.7	1.6	versicolor
57	4.9	2.4	3.3	1.0	versicolor
58	6.6	2.9	4.6	1.3	versicolor
59	5.2	2.7	3.9	1.4	versicolor
60	5.0	2.0	3.5	1.0	versicolor
61	5.9	3.0	4.2	1.5	versicolor
62	6.0	2.2	4.0	1.0	versicolor
63	6.1	2.9	4.7	1.4	versicolor
64	5.6	2.9	3.6	1.3	versicolor
65	6.7	3.1	4.4	1.4	versicolor
66	5.6	3.0	4.5	1.5	versicolor

✓ Lock vertical scroll

iris_str

	sepal_length String	sepal_width Numeric	petal_length Numeric	petal_width Numeric	species String
52	6.9	3.1	4.9	1.5	versicolor
53	5.5	2.3	4.0	1.3	versicolor
54	6.5	2.8	4.6	1.5	versicolor
55	5.7	2.8	4.5	1.3	versicolor
56	6.3	3.3	4.7	1.6	versicolor
57	4.9	2.4	3.3	1.0	versicolor
58	6.6	2.9	4.6	1.3	versicolor
59	5.2	2.7	3.9	1.4	versicolor
60	5	2.0	3.5	1.0	versicolor
61	5.9	3.0	4.2	1.5	versicolor
62	6	2.2	4.0	1.0	versicolor
63	6.1	2.9	4.7	1.4	versicolor
64	5.6	2.9	3.6	1.3	versicolor
65	6.7	3.1	4.4	1.4	versicolor
66	5.6	3.0	4.5	1.5	versicolor

Figure 10: Comparison of two dataframes side by side

inside the `logs/ops` folder. In this case every program session initialises a new log file.
Additionally debug information as well as critical errors are logged inside `logs/app`. These logs can be useful to debug software crashes and unexpected behaviours.
The log directory can be opened using the predefined window manager from menu `Help > "Open log directory"`.
Finally, option `Help > "Delete old logs"` clean the log directory removing all but the 5 most recent files.

2.7 Extraction of a time series: an example

Date	Births
1959-01-01	35
1959-01-02	32
1959-01-03	30
⋮	⋮
1959-12-30	55
1959-12-31	50

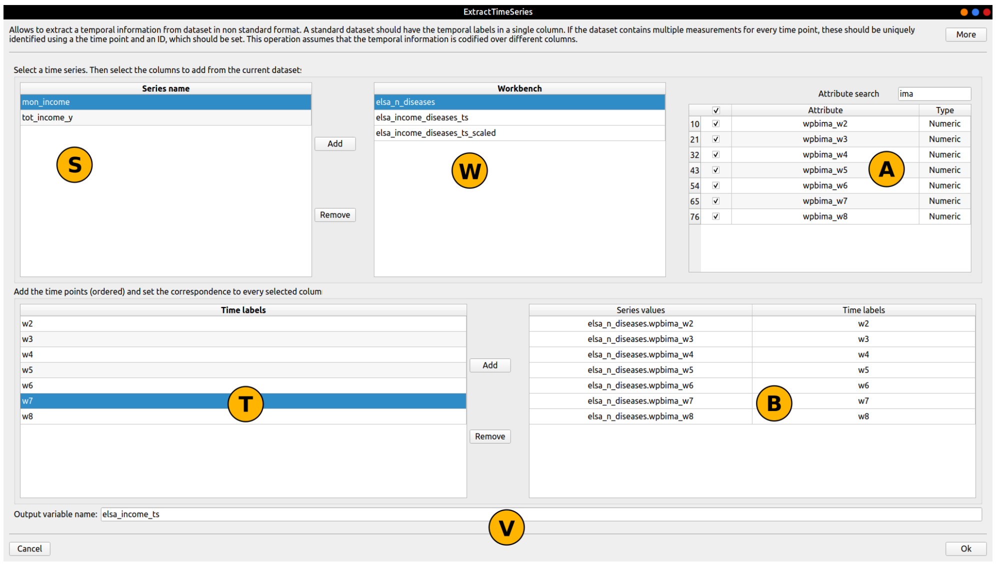
Table 1: Some rows from the *daily-total-female-births* dataset

idauniq	wpwlyy	wpbima	...
121321	...	...	
121323	...	...	
121332	...	...	
⋮			

Table 2: Structure of a wave of the ELSA dataset

The dataset used to plot the time series in Fig. 7 is composed of 365 entries, each with a date (under column *Date*) and the number of female births on that day (column *Births*). A sample of this dataset is reported in Table 1. The *View panel* can understand dataset in this simple format: it is only necessary to convert the *Date* attribute to *datetime*, set it as the time axis and select the time-dependent attributes, which is only *Births* in this example.
Longitudinal dataset like ELSA or HRS are not so simple: first there is no attribute representative of the time axis, but rather this information is implicitly conveyed by putting the variables from different waves in distinct files, or, if they are all in the same file, by renaming the variables to reference the wave they belong to (e.g. *varA_wave1*, *varA_wave2*, etc.). Hence, with these datasets the temporal information must be made explicit, by defining a time axis and, with the user help, link every column of the datasets to their point in this axis. For instance the file containing ELSA data from wave 3 has the structure described in Table 2. For the purpose of this example, only 3 attributes are listed: the *idauniq* attribute contains the unique cross-wave identifier, *wpwlyy* is the amount of gross income gained from the respondent work at the end of the previous year and *wpbima* is the average monthly income

from business during the previous 12 months (from the interview). Other waves follow the same scheme, even though variable names can sometimes change across different waves. Suppose we want to visualise how the respondent income changes from wave



Allows to extract a temporal information from dataset in non standard format. A standard dataset should have the temporal labels in a single column. If the dataset contains multiple measurements for every time point, these should be uniquely identified using a the time point and an ID, which should be set. This operation assumes that the temporal information is codified over different columns.

Select a time series. Then select the columns to add from the current dataset:

Series name
mon_income
tot_income_y

Workbench
elsa_n_diseases
elsa_income_diseases_ts
elsa_income_diseases_ts_scaled

Attribute search: ima

	✓	Attribute	Type
10	✓	wpbima_w2	Numeric
21	✓	wpbima_w3	Numeric
32	✓	wpbima_w4	Numeric
43	✓	wpbima_w5	Numeric
54	✓	wpbima_w6	Numeric
65	✓	wpbima_w7	Numeric
76	✓	wpbima_w8	Numeric

Add the time points (ordered) and set the correspondence to every selected colum

Time labels
w2
w3
w4
w5
w6
w7
w8

Series values	Time labels
elsa_n_diseases.wpbima_w2	w2
elsa_n_diseases.wpbima_w3	w3
elsa_n_diseases.wpbima_w4	w4
elsa_n_diseases.wpbima_w5	w5
elsa_n_diseases.wpbima_w6	w6
elsa_n_diseases.wpbima_w7	w7
elsa_n_diseases.wpbima_w8	w8

Output variable name: elsa_income_ts

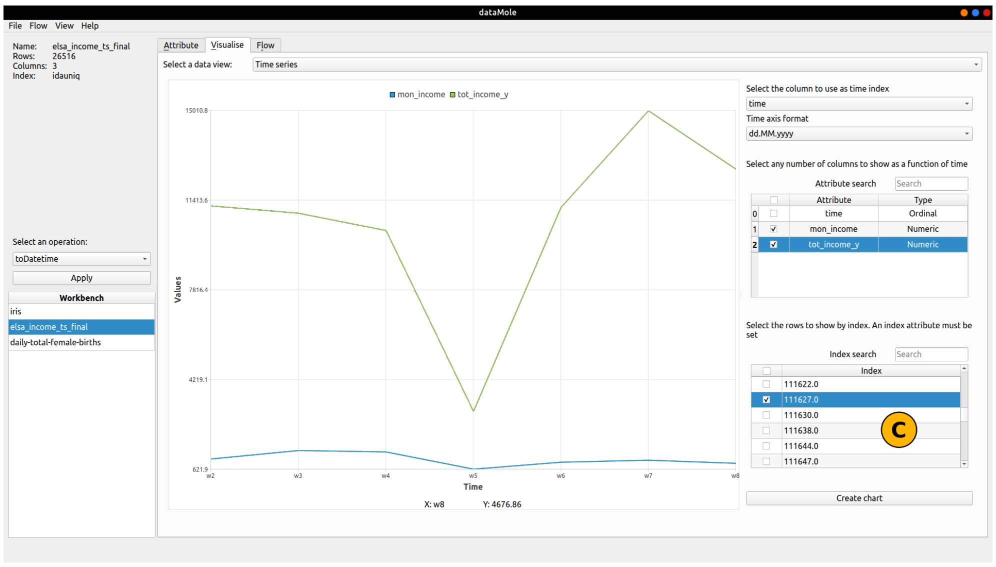
Figure 11: Extraction of time series from ELSA attribute *wpbima*

2 to 8 and plot it as a time series: there is no immediate way to do it, and we have to manipulate this dataset and transform it into a time series that can be identified and used in the *View panel*. The *time series extraction* operation is designed for this use case. Fig. 11 shows the widget used to configure the operation. In this case the relevant attributes (*wpbima* and *wpwlyy*) were previously extracted from their files and merged together in a single dataframe using the *join* operation. They were also renamed by appending the name of the waves they referred to (*_w2* for the second wave, *_w3* for the third and so on). This attributes contain a majority of NaN values: for the purpose of this example, they were filled with the per-column average, even though, in a real use case, they would probably be unusable.
In Fig. 11 two time series are defined and added to list *(S)*: *mon_income* for attribute *wpbima*, and *tot_income_y* for *wpwlyy*.
Then every time series is populated with its values: since we are interested in the average monthly income from the second to the eight wave, attributes from *wpbima_w2* to *wpbima_w8* are selected in table *(A)*. Here all attributes are taken from a single dataframe, but it is generally possible to select attributes from different dataframes by choosing the right ones in the workbench shown in section *(W)*.
The next step is the association of these attributes to a time point (the wave in this case). Thus, after defining an appropriate number of labels for the time axis in the bottom-left table *(T)*, we associate every relevant column to its time label in the bottom-right table *(B)*: here we are basically telling DataMole that *wpbima_w2* contains the values of the attribute *wpbima* for wave 2, *wpbima_w3* contains the values for wave 3 and so on. Series *tot_income_y* is built in the same way, by selecting the attributes *wpwlyy* from the various waves. Finally, a name for the dataset is set in *(V)* and the operation is started.
The result of this operation is a new dataset with the structure shown in Table 3.

idauniq	time	tot_income_y	mon_income
121321	w2	...	...
	w3	...	...
	w4	...	...
	w5	...	...
	w6	...	...
	w7	...	...
	w8	...	...
121323	w2	...	...
	w3	...	...
	w4	...	...
	w5	...	...
	w6	...	...
	w7	...	...
	w8	...	...
121332	w2	...	...
	w3	...	...
	w4	...	...
	w5	...	...
	w6	...	...
	w7	...	...
	w8	...	...
⋮			

Table 3: Structure of the dataset generated with the extraction of two time series (*tot_income_y* and *mon_income*) from the ELSA dataset

Every respondent is associated to 7 entries, each containing a distinct time label (*time* attribute) and the values of the *wpwlyy* and *wpbima* columns, respectively under columns *tot_income_y* and *mon_income*.
At this point it is possible to plot these attributes: you can see how the average monthly income and the yearly income changes for a single respondent (Fig. 12) or you can compare the same attribute (yearly income or monthly income) from two different respondents (Fig. 13). In both cases the table on the bottom (named *(C)* in Fig. 12) must be used to select the ids of the respondents to plot. This table allows to select by dataframe index, that must be set appropriately: in this example it is set on the respondent id. Additionally when an index is selected, the chart expects to find a time attribute and some time-dependent attributes to plot for every selected index, exactly like in Table 3. Hence table *(C)* should be used only to plot time series extracted with the *time series extraction* operation.



Name:	elsa_income_ts_final
Rows:	26516
Columns:	3
Index:	idauniq

Select an operation: toDatetime

Select a data view: Time series

Select the column to use as time index: time

Time axis format: dd.MM.yyyy

Select any number of columns to show as a function of time

Attribute search: Search

		Attribute	Type
0		time	Ordinal
1	✓	mon_income	Numeric
2	✓	tot_income_y	Numeric

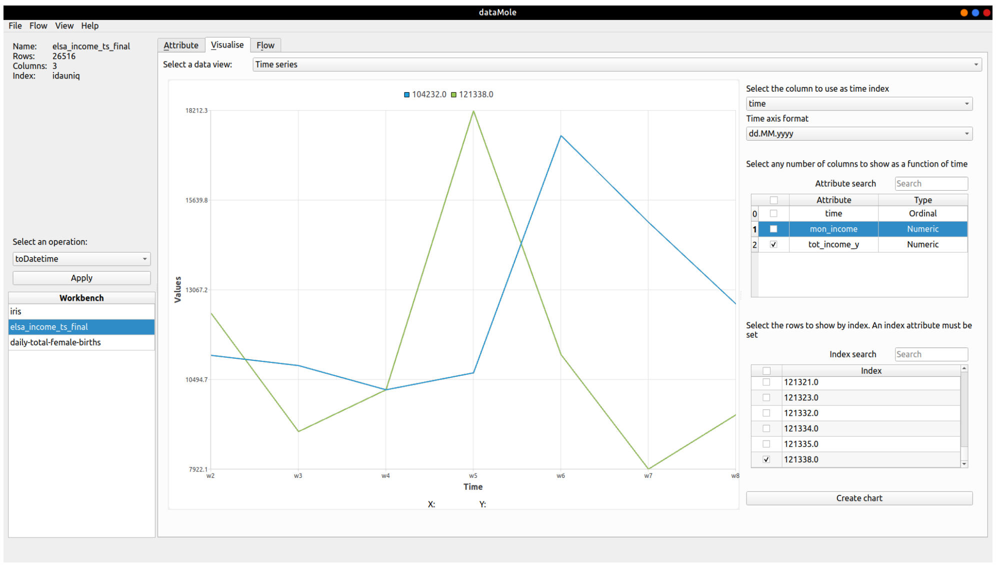
Select the rows to show by index. An index attribute must be set

Index search: Search

	Index
	111622.0
✓	111627.0
	111630.0
	111638.0
	111644.0
	111647.0

X: w8 Y: 4676.86

Figure 12: The average monthly income and yearly income of a single ELSA respondent measured in wave 2 to 8 (ELSA attributes *wpbima* and *wpwlyy*). The time labels for every wave are shown on the *Time* axis (horizontal), while the vertical axis shows the values of the two selected attributes.



Name:	elsa_income_ts_final
Rows:	26516
Columns:	3
Index:	idauniq

Select an operation: toDatetime

Select a data view: Time series

Select the column to use as time index: time

Time axis format: dd.MM.yyyy

Select any number of columns to show as a function of time

Attribute search: Search

		Attribute	Type
0		time	Ordinal
1		mon_income	Numeric
2	✓	tot_income_y	Numeric

Select the rows to show by index. An index attribute must be set

Index search: Search

	Index
	121321.0
	121323.0
	121332.0
	121334.0
	121335.0
✓	121338.0

X: Y:

Figure 13: The yearly income of two ELSA respondents from wave 2 to 8 (ELSA attribute *wpwlyy*)