ADAMS

Advanced **D**ata mining **A**nd **M**achine learning **S**ystem

Module: adams-spreadsheet

Peter Reutemann

January 10, 2024

©2012-2022

THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

Except where otherwise noted, this work is licensed under
http://creativecommons.org/licenses/by-sa/4.0/

Contents

List of Figures

Chapter 1

Introduction

Tabular data is a very common data format, not only for machine learning. The *spreadsheet* module offers some basic spreadsheet support for reading/writing and some generic actors. Other modules, like the *odf* or *excel* one, offer other native readers and writers. The data read by these readers can be processed with the same actors.

Chapter 2

Flow

A lot of the following actors and conversion schemes offer either an index of a column or a range of columns as a parameter. In addition to the usual *first/last/...* placeholders, you can use the actual column names (case-insensitive). This makes an actor less error-prone, in case the order of the columns may change.

The following standalones are available:

- *LookUpInit* – Initializes an empty lookup table in storage, which needs populating using the *LookUpAdd* transformer.[1]

The following sources are available:

- *DatabaseMetaData* – outputs spreadsheets with information obtained from the meta-data of a database connection.
- *NewSpreadSheet* – for creating an empty spreadsheet with pre-defined columns.
- *SpreadSheetDbReader* – turns results from SQL queries into spreadsheet objects.[2]
- *LookUp* – Outputs a value from a stored lookup table, identified by a user-supplied key.[3]
- *LookUpTable* – Outputs a stored lookup table as spreadsheet.[4]

The following transformers are available:

- *LookUp* – obtains the value associated with the string received as input from an internally stored lookup table.[5]
- *LookUpAdd* – adds a key/value pair to a lookup table.[6]
- *LookUpInit* – initializes a lookup table by using two columns from a spreadsheet, one acting as key, the other as value.[7]
- *LookUpRemove* – removes a key/value pair from a lookup table.[8]

[1]adams-spreadsheet-lookup2.flow
[2]adams-spreadsheet-database_access.flow
[3]adams-spreadsheet-lookup.flow
[4]adams-spreadsheet-lookup.flow
[5]adams-spreadsheet-lookup.flow
[6]adams-spreadsheet-lookup.flow
[7]adams-spreadsheet-lookup.flow
[8]adams-spreadsheet-lookup.flow

- *MakeJFreeChartDataset* – generates a JFreeChart[2] dataset from a spreadsheet.
- *MultiSpreadSheetOperation* – applies the specified operation to the incoming spreadsheet array (e.g., calculating difference, merging, etc)
- *SpreadSheetAggregate* – aggregates rows in a spreadsheet (min, max, average, standard deviation, etc).[9]
- *SpreadSheetAnonymize* – for anonymizing columns.[10]
- *SpreadSheetCellFinder* – outputs row/column pairs of cells that matched the specified criteria of the cell finder algorithm.
- *SpreadSheetCellSelector* – lets the user select cells in a spreadsheet interactively to be forwarded in the flow.
- *SpreadSheetCollapse* – collapses cell values of rows with the same key into single cell.
- *SpreadSheetColumnFilter* – filters columns using a column finder scheme.[11]
- *SpreadSheetColumnIterator* – iterates over all the columns in the spreadsheet and outputs the names.[12]
- *SpreadSheetColumnStatistic* – generates statistics for a spreadsheet column.[13]
- *SpreadSheetColumnsByName* – generates a new spreadsheet with only the columns that match a regular expression (inverting is possible as well).
- *SpreadSheetCommonIDs* – determines common (or not in common) IDs from two or more spreadsheets.
- *SpreadSheetConvertCells* – applies arbitrary conversion schemes to individual cells.
- *SpreadSheetConvertHeaderCells* – applies arbitrary conversion schemes to the header cells.
- *SpreadSheetCopyColumns* – copies the content of a range of columns into new columns.
- *SpreadSheetCopyRows* – duplicates the content of a range of rows at another location in the spreadsheet.[14]
- *SpreadSheetDifference* – computes the difference between two spreadsheets.
- *SpreadSheetExtractArray* – allows the extraction of a row or column from a spreadsheet.
- *SpreadSheetFileReader* – for reading spreadsheet files; depending on the reader, multiple sheets can get read at once.[15]
- *SpreadSheetFilter* – applies the specified filter to the spreadsheet.
- *SpreadSheetGetCell* – retrieves the value of a specific cell in the spreadsheet.[16]
- *SpreadSheetGetColumnIndex* – retrieves the 1-based index columns which name matches a regular expression.[17]

[9] adams-spreadsheet-aggregate.flow
[10] adams-spreadsheet-anonymize_columns.flow
[11] adams-spreadsheet-filter_columns.flow
[12] adams-spreadsheet-iterate_cols.flow
[13] adams-spreadsheet-column_statistics.flow
[14] adams-spreadsheet-copy_rows.flow
[15] adams-spreadsheet-output_cells.flow
[16] adams-spreadsheet-output_cells.flow
[17] adams-spreadsheet-get_column_index.flow

- *SpreadSheetGetHeaderCell* – retrieves the value of a specific cell in the header row of a spreadsheet.
- *SpreadSheetHistogramRanges* – outputs the ranges generated by the ArrayHistogram statistic.
- *SpreadSheetInfo* – generates basic information on the spreadsheet object.
- *SpreadSheetInsertColumn* – inserts a column in the spreadsheet, initializes the cells with a user-defined value.
- *SpreadSheetInsertRow* – inserts a row in the spreadsheet, initializes the cells with a user-defined value.
- *SpreadSheetInsertRowScore* – inserts a column in the spreadsheet, containing a score calculate for the corresponding row.
- *SpreadSheetMatrixStatistic* – calculates statistics from the whole spreadsheet (or a defined subset).
- *SpreadSheetMerge* – merges multiple spreadsheets into a single one.[18]
- *SpreadSheetMethodMerge* – like *SpreadSheetMerge*, merges multiple spreadsheets into a single, but uses a class hierarchy of merge algorithms.
- *SpreadSheetPlotGenerator* – turns a spreadsheet into plot containers to be displayed in the *SequencePlotter* sink.[19]
- *SpreadSheetQuery* – allows to run an SQL-like query on a spreadsheet to select a subset of rows/columns, delete rows, rename columns, update cells.[20]
- *SpreadSheetRandomSystematicSample* – picks a random, systematic sample of rows from a spreadsheet.
- *SpreadSheetRemoveColumn* – removes columns from a spreadsheet.
- *SpreadSheetRemoveRow* – removes rows from a spreadsheet.
- *SpreadSheetReorderColumns* – reorders columns in a spreadsheet, also allows duplicating/dropping of columns.
- *SpreadSheetReorderRows* – reorders rows in a spreadsheet, also allows duplicating/dropping of rows.
- *SpreadSheetReplaceCellValue* – replaces cell values that match a regular expression.
- *SpreadSheetRowBinning* – adds a column with the bin index calculated by a binning algorithm from the numeric values of a column in the spreadsheet.
- *SpreadSheetRowBuffer* – buffers incoming row objects and outputs spreadshets or outputs single rows when receiving spreadsheets.
- *SpreadSheetRowFilter* – filters rows using a row finder scheme.[21]
- *SpreadSheetRowStatistic* – generates statistics for a spreadsheet row.[22]
- *SpreadSheetSelectSubset* – allows the user to select a subset from a spreadsheet.
- *SpreadSheetSetCell* – sets the value of a specific cell in the spreadsheet.[23]

[18] adams-spreadsheet-simple_merge.flow, adams-spreadsheet-merge_using_id.flow

[19] adams-spreadsheet-spreadsheet_plot1.flow, adams-spreadsheet-spreadsheet_plot2.flow, adams-spreadsheet-statistic.flow

[20] adams-spreadsheet-query.flow, adams-spreadsheet-date_queries.flow, adams-spreadsheet-celltype_queries.flow

[21] adams-spreadsheet-filter_rows.flow

[22] adams-spreadsheet-row_statistics.flow

[23] adams-spreadsheet-set_cells.flow

- *SpreadSheetSetHeaderCell* – sets the value of a specific cell in the header row of a spreadsheet.
- *SpreadSheetSort* – sorts the rows a spreadsheet using an arbitrary number of columns (ascending or descending).
- *SpreadSheetSortColumns* – sorts the columns (or a subset of columns) of a spreadsheet using a specified comparator.
- *SpreadSheetStatistic* – calculates statistics using the data stored in the spreadsheet.[24]
- *SpreadSheetSubset* – for obtaining a subset of the spreadsheet object (subset of columns and/or rows).
- *SpreadSheetSubsetByValue* – splits a spreadsheet into subsets using the unique (string) values in a column for grouping.
- *SpreadSheetSubsetFromGroup* – retrieves the specified rows from groups within a spreadsheet (group as determined by a sorted column).
- *SpreadSheetSupporterToSpreadSheet* – converts an object that implements SpreadSheetSupporter into the corresponding spreadsheet representation.
- *SpreadSheetStorageRowIterator* – iterates over a range of rows/columns and stores the cell values in internal storage.
- *SpreadSheetTransformCells* – applies an arbitrary callable transformer to individual cells.
- *SpreadSheetTransformHeaderCells* – applies an arbitrary callable transformer to the header cells.
- *SpreadSheetVariableRowIterator* – iterates over a range of rows/columns and stores the cell values in variables.[25]
- *StorageJFreeChartAddSeries* – generates a JFreeChart[2] series from the spreadsheet and adds it to the specified dataset in storage.

The following sinks are available:

- *JFreeChartFileWriter* – generates plots from spreadsheet columns or a JFreeChart[2] dataset and writes them to disk.
- *JFreeChartPlot* – allows plotting of columns from spreadsheets or a JFreeChart[2] dataset.
- *SpreadSheetDbWriter* – for storing a spreadsheet in a database.[26]
- *SpreadSheetDisplay* – for displaying spreadsheet objects in tabular form.[27]
- *SpreadSheetFileWriter* – writes spreadsheet objects to a file with the chosen writer class; depending on the writer either a single or multiple sheets can get written at once.
- *SpreadSheetRowViewer* – displays the numeric values of rows from a spreadsheet as line plots.

The following boolean conditions are available:

- *HasColumn* – checks whether the spreadsheet passing through has a column with that name.

The following conversion schemes are available:

[24] adams-spreadsheet-statistic.flow
[25] adams-spreadsheet-variable_row_iterator.flow
[26] adams-spreadsheet-database_access.flow
[27] adams-spreadsheet-display.flow

- *ContainerToSpreadSheet* – for converting any flow container (e.g., a prediction container) into a spreadsheet for better visualization.
- *ConvertSpreadSheetRows* – converts the data rows in a spreadsheet into a different format, e.g., into sparse representation.
- *DoubleMatrixToSpreadSheet* – converts a two-dimensional double array (i.e., matrix) into a spreadsheet object.
- *MapToSpreadSheet* – turns a *java.util.Map* object into a spreadsheet.
- *NotesToSpreadSheet* – generates a spreadsheet from a Notes object.
- *PropertiesToSpreadSheet* – generates a spreadsheet from a Properties object.
- *RenameSpreadSheet* – renames a spreadsheet.
- *RenameSpreadSheetColumn* – renames a single column in a spreadsheet.
- *ReportToSpreadSheet* – generates a spreadsheet from a Report object.
- *RowArrayToSpreadSheet* – generates a spreadsheet from an array of spreadsheet rows.
- *SpreadSheetAddFormulaColumn* – adds a column with a row-wise, user-supplied formula.
- *SpreadSheetAddFormulaRow* – adds a row with a column-wise, user-supplied formula.
- *SpreadSheetAddRowID* – adds an ID column to the spreadsheet that contains the row index as value
- *SpreadSheetAddSumColumn* – adds a column with a row-wise sum formula.
- *SpreadSheetAddSumRow* – adds a row with a column-wise sum formula.
- *SpreadSheetAnyColumnToString* – converts any data types in a column to strings.
- *SpreadSheetCellLocationToCoordinates* – turns a cell location into an integer array of 1-based coordinates
- *SpreadSheetCellLocationToPosition* – turns a cell location obtained from a cell finder into a position string (eg 'A2').
- *SpreadSheetColumnFinderToRange* – outputs a range string of the columns that the finder located (eg '1-6,8').
- *SpreadSheetColumnsToReport* – converts columns in a spreadsheet to Report objects.
- *SpreadSheetDoubleColumnToLong* – converts any floating values to integer (= long) values.
- *SpreadSheetDoubleColumnToString* – converts any floating values to strings.
- *SpreadSheetEscapeColumnName* – ensures that a column is escaped correctly in order to be used in column range expressions.
- *SpreadSheetInsertCellLocation* – replaces a placeholder in a string with a cell location (e.g., “A1”).
- *SpreadSheetInsertColumnPosition* – replaces a placeholder in a string with a column position (e.g., “BG“)
- *SpreadSheetJoinColumns* – joins two or more columns in a spreadsheet into a single one.
- *SpreadSheetLongColumnToDouble* – converts long values in a column to double objects.

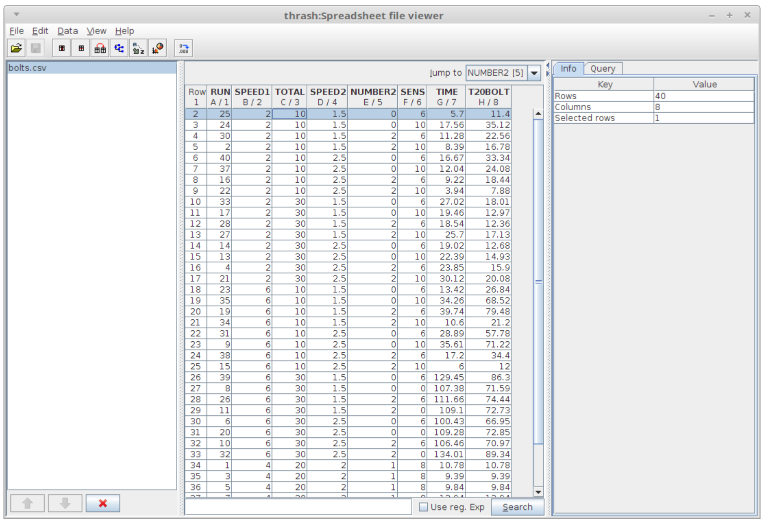
- *SpreadSheetMaterializeFormulas* – replaces formulas in cells with the current value of the formula.
- *SpreadSheetRowFinderToRange* – outputs a range string of the rows that the finder located (eg '1-6,8').
- *SpreadSheetRowsToReport* – converts rows in a spreadsheet to Report objects.
- *SpreadSheetStringSplitColumn* – splits the string representation of a column into multiple columns using a regular expression.
- *SpreadSheetStringColumnToBoolean* – converts strings values in a column to boolean objects.
- *SpreadSheetStringColumnToDate* – converts strings values in a column to date objects.
- *SpreadSheetStringColumnToDateTime* – converts strings values in a column to date/time objects.
- *SpreadSheetStringColumnToDouble* – converts strings values in a column to floating values.
- *SpreadSheetStringColumnToLong* – converts strings values in a column to integer (= long) values.
- *SpreadSheetStringColumnToTime* – converts strings values in a column to time objects.
- *SpreadSheetToCreateTableStatement* – turns a spreadsheet with SQL column names and types into a SQL 'CREATE TABLE' statement.
- *SpreadSheetToDoubleMatrix* – turns all the numeric columns of a spreadsheet into a two-dimensional double matrix.
- *SpreadSheetToMap* – uses two columns from a spreadsheet (key and value) to populate a *java.util.Map* object.
- *SpreadSheetToNumeric* – turns all non-numeric cells into numeric ones; can replace missing values as well.
- *SpreadSheetToRowArray* – turns a spreadsheet into an array of spreadsheet rows.
- *SpreadSheetToStringMatrix* – turns all columns of a spreadsheet into a two-dimensional string matrix.
- *SpreadSheetUnescapeColumnName* – reverses the escaping for a column name.
- *SpreadSheetUniqueColumnNames* – ensures that the column names uniquely identify a column.
- *SpreadSheetUseRowAsHeader* – replaces the header values with the ones from the specified data row.
- *StringToSpreadSheet* – parses a string in CSV format and turns it into a spreadsheet object.
- *StringMatrixToSpreadSheet* – turns a (two-dimensional) string matrix into a spreadsheet.
- *TransposeSpreadSheet* – swaps columns with rows.

Chapter 3

Tools

3.1 Spreadsheet file viewer

The *Spreadsheet file viewer* is a simple tool for loading all spreadsheet file formats that ADAMS supports. Despite its name, the tool also allows you to modify cell values and save them back to a file. Figure 3.1 shows a dataset that was loaded from a CSV (comma-separated values) file.



Row 1	RUN A / 1	SPEED1 B / 2	TOTAL C / 3	SPEED2 D / 4	NUMBER2 E / 5	SENS F / 6	TIME G / 7	T20BOLT H / 8
2	25	2	10	1.5	0	6	5.7	11.4
3	24	2	10	1.5	0	10	17.56	35.12
4	30	2	10	1.5	2	6	11.28	22.56
5	2	2	10	1.5	2	10	8.39	16.78
6	40	2	10	2.5	0	6	16.67	33.34
7	37	2	10	2.5	0	10	12.04	24.08
8	16	2	10	2.5	2	6	9.22	18.44
9	22	2	10	2.5	2	10	3.94	7.88
10	33	2	30	1.5	0	6	27.02	18.01
11	17	2	30	1.5	0	10	19.46	12.97
12	28	2	30	1.5	2	6	18.54	12.36
13	27	2	30	1.5	2	10	25.7	17.13
14	14	2	30	2.5	0	6	19.02	12.68
15	13	2	30	2.5	0	10	22.39	14.93
16	4	2	30	2.5	2	6	23.85	15.9
17	21	2	30	2.5	2	10	30.12	20.08
18	23	6	10	1.5	0	6	13.42	26.84
19	35	6	10	1.5	0	10	34.26	68.52
20	19	6	10	1.5	2	6	39.74	79.48
21	34	6	10	1.5	2	10	10.6	21.2
22	31	6	10	2.5	0	6	28.89	57.78
23	9	6	10	2.5	0	10	35.61	71.22
24	38	6	10	2.5	2	6	17.2	34.4
25	15	6	10	2.5	2	10	6	12
26	39	6	30	1.5	0	6	129.45	86.3
27	8	6	30	1.5	0	0	107.38	71.59
28	26	6	30	1.5	2	6	111.66	74.44
29	11	6	30	1.5	2	0	109.1	72.73
30	6	6	30	2.5	0	6	100.43	66.95
31	20	6	30	2.5	0	0	109.28	72.85
32	10	6	30	2.5	2	6	106.46	70.97
33	32	6	30	2.5	2	0	134.01	89.34
34	1	4	20	2	1	8	10.78	10.78
35	3	4	20	2	1	8	9.39	9.39
36	5	4	20	2	1	8	9.84	9.84

Key	Value
Rows	40
Columns	8
Selected rows	1

Figure 3.1: Viewer for spreadsheet files.

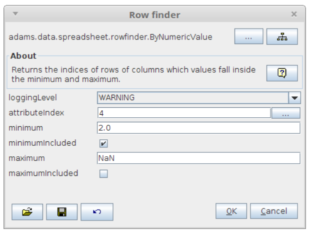
If there are more spreadsheet file formats registered, you can save the currently displayed spreadsheet in another format. Printing, of course, is available throught the *Send to* menu. By default, the viewer displays each cell with as many digits after the decimal point as necessary. But you can also unify this and specify how many digits should be used for all floating point cells.

3.1.1 Filtering and processing

The viewer supports some basic filtering and processing:

- *columns* – creates a subset of the spreadsheet by selecting a subset of columns, e.g., all columns which name starts with a certain string.
- *rows* – creates a subset of the spreadsheet by selecting a subset of rows, e.g., rows with a certain value in a column.
- *convert* – applies a conversion scheme specific to spreadsheets, e.g., transposing a spreadsheet.
- *transform* – applies a flow transformer specific to spreadsheets, e.g., inserting a column or creating a subset.

Figures 3.2 and 3.3 show the setup for a row finder filter and the resulting new spreadsheet.



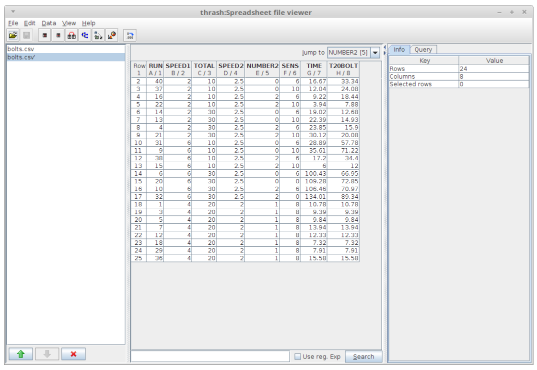
adams.data.spreadsheet.rowfinder.ByNumericValue

About

Returns the indices of rows of columns which values fall inside the minimum and maximum.

loggingLevel	WARNING
attributeIndex	4
minimum	2.0
minimumIncluded	✓
maximum	NaN
maximumIncluded	☐

Figure 3.2: Row finder setup.



Row 1	RUN A / 1	SPEED1 B / 2	TOTAL C / 3	SPEED2 D / 4	NUMBER2 E / 5	SENS F / 6	TIME G / 7	T20BOLT H / 8
2	40	2	10	2.5	0	6	16.67	33.34
3	37	2	10	2.5	0	10	12.04	24.08
4	16	2	10	2.5	2	6	9.22	18.44
5	22	2	10	2.5	2	10	3.94	7.88
6	14	2	30	2.5	0	6	19.02	12.68
7	13	2	30	2.5	0	10	22.39	14.93
8	4	2	30	2.5	2	6	23.85	15.9
9	21	2	30	2.5	2	10	30.12	20.08
10	31	6	10	2.5	0	6	28.89	57.78
11	9	6	10	2.5	0	10	35.61	71.22
12	38	6	10	2.5	2	6	17.2	34.4
13	15	6	10	2.5	2	10	6	12
14	6	6	30	2.5	0	6	100.43	66.95
15	20	6	30	2.5	0	0	109.28	72.85
16	10	6	30	2.5	2	6	106.46	70.97
17	32	6	30	2.5	2	0	134.01	89.34
18	1	4	20	2	1	8	10.78	10.78
19	3	4	20	2	1	8	9.39	9.39
20	5	4	20	2	1	8	9.84	9.84
21	7	4	20	2	1	8	13.94	13.94
22	12	4	20	2	1	8	12.33	12.33
23	18	4	20	2	1	8	7.32	7.32
24	29	4	20	2	1	8	7.91	7.91
25	36	4	20	2	1	8	15.58	15.58

Key	Value
Rows	24
Columns	8
Selected rows	0

Figure 3.3: The filtered spreadsheet.

3.1.2 Plug-ins

The viewer can be extended with two sorts of plug-ins:

- ones that generate a view based on the current sheet ("view")
- ones that process the current sheet ("data")

3.1.2.1 View plug-ins

A view plug-in is derived from the following super-class:

```
adams.gui.tools.spreadsheetviewer.AbstractViewPlugin
```

There are three methods that need implementing:

- *getMenuText()* – returns the text used for the menu item and the title of the dialog displaying the generated view.
- *getMenuIcon()* – returns the name of the icon (no path) that should be displayed in the menu (use null to display no icon).
- *doGenerate(SpreadSheet)* – this method generates the actual view in form of a *adams.gui.core.BasePanel*.

An example is the *Statistics* plug-in, which shows simple statistics for a spreadsheet, number of rows and colums and what types of columns are present:

```
adams.gui.tools.spreadsheetviewer.Statistics
```

Further superclasses:

- *AbstractSelectedSheetsViewPlugin* – for view plugins that operate on one or more spreadsheets that the user selects.
- *AbstractSelectedSheetsViewPluginWithGOE* – same as *AbstractSelectedSheetsViewPlugin* but offers the user to change the settings through a GenericObjectEditor view.

3.1.2.2 Data plug-ins

A view plug-in is derived from the following super-class:

```
adams.gui.tools.spreadsheetviewer.AbstractDataPlugin
```

There are four methods that need implementing:

- *getMenuText()* – returns the text used for the menu item and the title of the dialog displaying the generated view.
- *getMenuIcon()* – returns the name of the icon (no path) that should be displayed in the menu (use null to display no icon).
- *doProcess(SpreadSheet)* – this method processes the current spreadsheet and returns a new spreadsheet object.
- *isInPlace()* – returns whether the generated spreadsheet object should simply replace the current one ("in-place") or added as new tab.

Further superclasses:

- *AbstractSelectedSheetsDataPlugin* – for data plugins that operate on one or more spreadsheets that the user selects (see *Append* plugin).
- *AbstractSelectedSheetsDataPluginWithGOE* – same as *AbstractSelectedSheetsDataPlugin* but offers the user to change the settings through a GenericObjectEditor view (see *Merge* plugin).

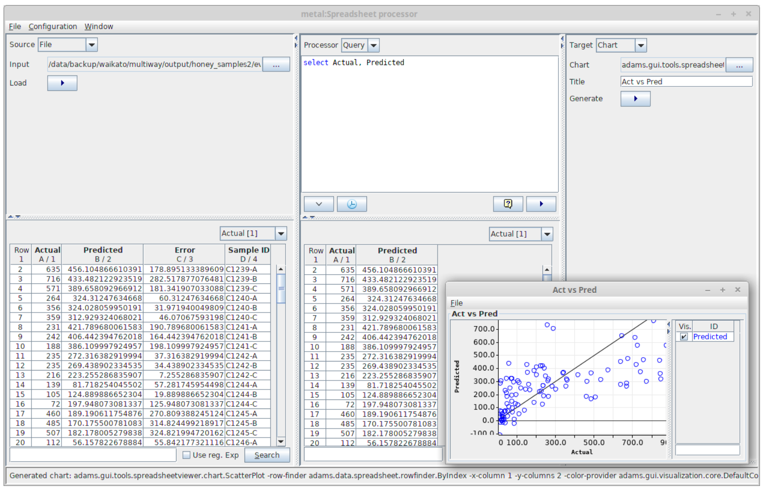
3.2 Spreadsheet processor

The *Spreadsheet processor* falls, in terms of functionality for handling spreadsheets, between the Preview browser and the Flow editor. It is mainly aimed at making it easier to post-process/clean or plot spreadsheets, without having to write a full-blown flow. It consists of three parts:

- *Source* – where to obtain the spreadsheet from (paste from clipboard, load from database, select specific file, file browser)
- *Processor* – how to process the spreadsheet (spreadsheet query, flow)
- *Target* – what to do with the processed spreadsheet (copy to clipboard, save to file, generated chart)

The tool also allows you to save configurations (source, processor and target), to be reloaded at a later stage to avoid having to re-setup the processing pipeline.

Figure 3.4 shows a model evaluation dataset that was loaded from a CSV (comma-separated values) file, two columns selected using the *spreadsheet query* processor and then displayed as chart (scatter plot with a diagonal).



Source: File

Input: /data/backup/waikato/multiway/output/honey_samples2/ev

Processor: Query

```
select Actual, Predicted
```

Target: Chart

Chart: adams.gui.tools.spreadsheet

Title: Act vs Pred

Actual [1]

Row 1	Actual A / 1	Predicted B / 2	Error C / 3	Sample ID D / 4
2	635	456.104866610391	178.895133389609	C1239-A
3	716	433.482122923519	282.517877076481	C1239-B
4	571	389.658092966912	181.341907033088	C1239-C
5	264	324.31247634668	60.31247634668	C1240-A
6	356	324.028059950191	31.971940049809	C1240-B
7	359	312.929324068021	46.07067593198	C1240-C
8	231	421.789680061583	190.789680061583	C1241-A
9	242	406.442394762018	164.442394762018	C1241-B
10	188	386.109997924957	198.109997924957	C1241-C
11	235	272.316382919994	37.316382919994	C1242-A
12	235	269.438902334535	34.438902334535	C1242-B
13	216	223.255286835907	7.255286835907	C1242-C
14	139	81.718254045502	57.281745954498	C1244-A
15	105	124.889886652304	19.889886652304	C1244-B
16	72	197.948073081337	125.948073081337	C1244-C
17	460	189.190611754876	270.809388245124	C1245-A
18	485	170.175500781083	314.824499218917	C1245-B
19	507	182.178005279838	324.821994720162	C1245-C
20	112	56.157822678884	55.842177321116	C1246-A

Actual [1]

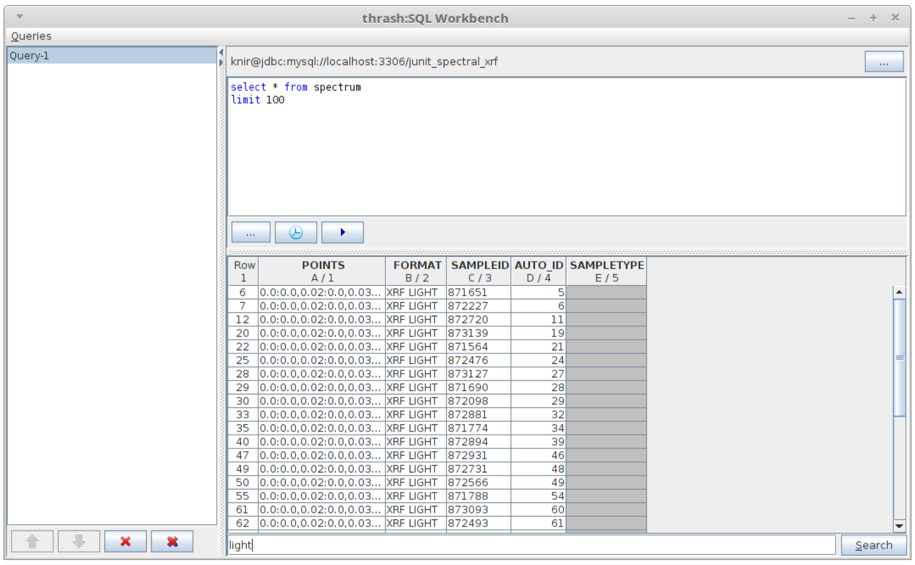
Row 1	Actual A / 1	Predicted B / 2
2	635	456.104866610391
3	716	433.482122923519
4	571	389.658092966912
5	264	324.31247634668
6	356	324.028059950191
7	359	312.929324068021
8	231	421.789680061583
9	242	406.442394762018
10	188	386.109997924957
11	235	272.316382919994
12	235	269.438902334535
13	216	223.255286835907
14	139	81.718254045502
15	105	124.889886652304
16	72	197.948073081337
17	460	189.190611754876
18	485	170.175500781083
19	507	182.178005279838
20	112	56.157822678884

Generated chart: adams.gui.tools.spreadsheetviewer.chart.ScatterPlot -row-finder adams.data.spreadsheet.rowfinder.ByIndex -x-column 1 -y-columns 2 -color-provider adams.gui.visualization.core.DefaultCo

Figure 3.4: Processing tool for spreadsheet files.

3.3 SQL Workbench

The *SQL Workbench* tool allows you to run queries (SELECT, UPDATE, DELETE) from within ADAMS, without having to rely on third-party database tools. You can have an arbitrary number of query panels and a history of your queries is kept as well. Further analysis or export of the results is possible via the result table's right-click popup menu.



knir@jdbc:mysql://localhost:3306/junit_spectral_xrf

```
select * from spectrum
limit 100
```

Row 1	POINTS A / 1	FORMAT B / 2	SAMPLEID C / 3	AUTO_ID D / 4	SAMPLETYPE E / 5
6	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	871651	5	
7	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	872227	6	
12	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	872720	11	
20	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	873139	19	
22	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	871564	21	
25	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	872476	24	
28	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	873127	27	
29	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	871690	28	
30	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	872098	29	
33	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	872881	32	
35	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	871774	34	
40	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	872894	39	
47	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	872931	46	
49	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	872731	48	
50	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	872566	49	
55	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	871788	54	
61	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	873093	60	
62	0.0:0.0,0.02:0.0,0.03...	XRF LIGHT	872493	61	

light

Figure 3.5: Workbench for SQL queries.

Chapter 4

Formulas

ADAMS supports formulas with a range of basic functions. The following lists describe briefly what functionality is available. A full list is available through the *Help -¿ Formulas* menu in the *Spreadsheet file viewer*.

Operands:

- $NUM + NUM$ – addition
- $NUM - NUM$ – subtraction
- $NUM * NUM$ – multiplication
- NUM/NUM – division
- $NUM \wedge NUM$ – exponential
- $NUM\%NUM$ – modulo

Boolean operations:

- $<$ – less than
- $<=$ – less than or equal
- $>$ – greater than
- $>=$ – greater than or equal
- $=$ – equals
- $! =$ – does not equal (alternative: $<>$)
- ! – negation
- & – and
- $|$ – or

Numeric functions:

- *abs* – absolute value of a cell/number.
- *average* – average computed from a range of cells.
- *ceil* – smallest value that is greater than or equal to the cell/number and is equal to a mathematical integer.
- *cos* – the trigonometric cosine of a number/cell.
- *countblank* – counts empty/missing value cells in a range of cells.
- *countif* – counts value only a condition is true.
- *exp* – Euler's number e raised to the power of a number/cell.

- *floor* – largest value that is less than or equal to the cell/number and is equal to a mathematical integer.
- *if[else]* – if-then-else construct.
- *intercept* – compute the intercept of linear regression between two cell ranges.
- *log* – natural logarithm (base e) of a number/cell.
- *max* – largest value from range of cells.
- *min* – smallest value from range of cells.
- *pow[er]* – the first number/cell raised to the power of the second number/cell.
- *rint* – returns number that is closest in value to the number/cell and is equal to a mathematical integer.
- *sin* – trigonometric sine of a number/cell.
- *slope* – compute the slope of linear regression between two cell ranges.
- *sqrt* – the correctly rounded positive square root of a number/cell.
- *stdev* – the sample standard deviation from a range of cells.
- *stdevp* – the population standard deviation from a range of cells.
- *sum* – the sum over a range of cells.
- *sumif* – the conditional sum over a range of cells.
- *tan* – trigonometric tangent of a number/cell.

String functions:

- *concatenate* – concatenates up to 5 strings.
- *find* – returns the location of a search string in a string.
- *left* – returns substring of specified length from the left of the string.
- *len[gth]* – the length of a string.
- *lower[case]* – converts string to a lowercase one.
- *matches* – matches a string against a regular expression.
- *mid* – returns substring of specified length from the specified positiojn in the string.
- *right* – returns substring of specified length from the right of the string.
- *replace* – replaces a substring at a specified position with a new string.
- *rept* – returns a string made up of X copies of the supplied string.
- *trim* – removes all leading and trailing whitespaces.
- *substr* – creates a substring from a string, given start/end position.
- *substiture* – replaces occurrences of a search string in a string with a new string.
- *upper[case]* – converts string to an uppercase one.

Date/time functions:

- *year* – extracts the year from a date/time cell.
- *month* – extracts the month from a date/time cell.
- *day* – extracts the day from a date/time cell.
- *hour* – extracts the hour from a date/time cell.
- *minute* – extracts the minute from a date/time cell.
- *second* – extracts the second from a date/time cell.

- *weekday* – extracts the weekday from a date/time cell (Sunday=1, Saturday=7).
- *weeknum* – extracts the week number from a date/time cell.

Chapter 5

Troubleshooting

5.1 Fractional times

Despite database systems supporting fractional times, i.e., times with fractional seconds (= milliseconds), the JDBC interface does not support this (at this stage). This results in fractions always getting set to 0.

However, JDBC drivers, like the MySQL one starting with 5.1.37, may support in some cases the sending of fractional seconds to the server where the may be subject to rounding (for MySQL use `sendFractionalSeconds=true|false` in the JDBC URL).

In order to avoid the truncating of fractions, you can do two things:

- Create your own queries and have them executed using the *ExecSQL* standalone - this bypasses the JDBC driver's checks for fractions.
- Turn the columns in the spreadsheet, when using the *SpreadSheetDbWriter*, into strings. They will get converted automatically, at least in the case of MySQL, automatically into fractional times again before being inserted. Either use the *SpreadSheetAnyColumnToString* conversion or use `concat("", column_name)` to return the column *column_name* as string from the database in the first place.

Bibliography

[1] *ADAMS* – Advanced Data mining and Machine learning System
`https://adams.cms.waikato.ac.nz/`

[2] *JFreeChart* – is a free 100% Java chart library that makes it easy for developers to display professional quality charts in their applications.
`http://www.jfree.org/jfreechart/`