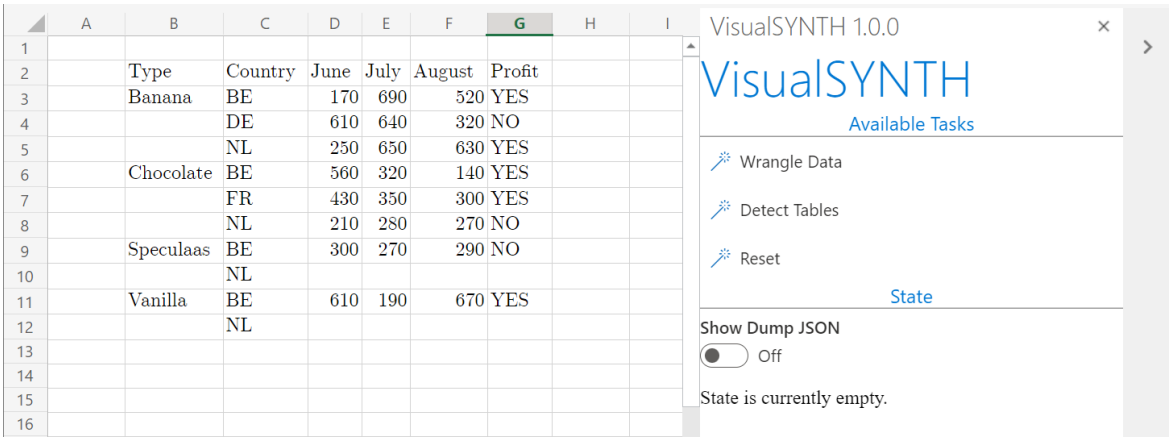
VisualSYNTH

a {visual} guide



Getting Started

Installation

Interacting with VisualSynth
- Sidebar
- Sketches

Formatting data

Cleaning data
- Detecting Errors
- Selecting Relevant Data

Constraints
- COUNTOR
- TaCLe

Predictions

Authors

Dr. Samuel Kolb

Dr. Clément Gautrias

Dr. Paolo Morettin

Dr. Mohit Kumar

Gust Verbruggen

Elia Van Wolputte

Nitesh Kumar

Sieben Bocklandt

Adem Kikaj

Getting Started

This manual provides an illustrated overview of the functionalities of **VisualSynth**.

With this tool, we want to integrate advanced data science tools and capabilities into the Microsoft Excel spreadsheet environment. In the spirit of democratizing data science, this tool is capable of solving many tasks semi-automatically, although some effort from the user is required. This effort is centered around which tasks the user wants to perform, rather than how they should be performed.

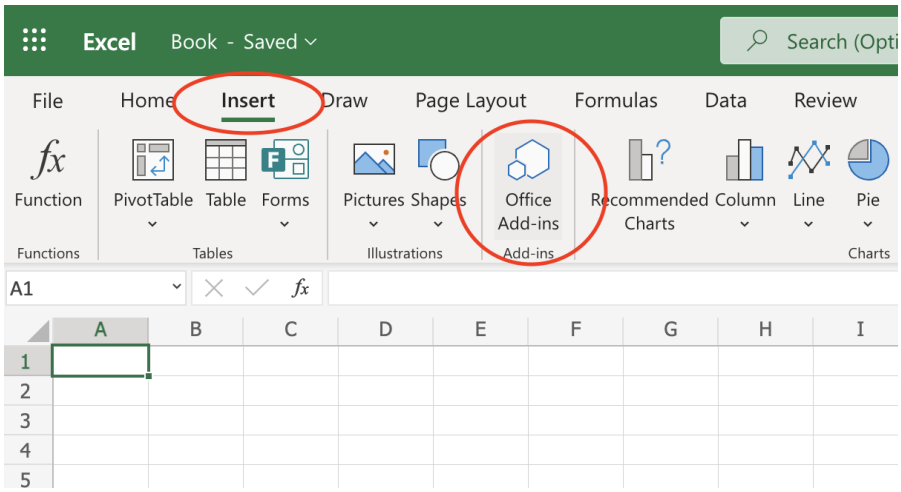
We will guide the user through a typical data science pipeline using some examples, starting from an unformatted spreadsheet.

Installation

To install the VisualSYNTH Add-in, first, you need to download the manifest.xml file (download here), which gets connected remotely to the services offered by the SYNTH software.

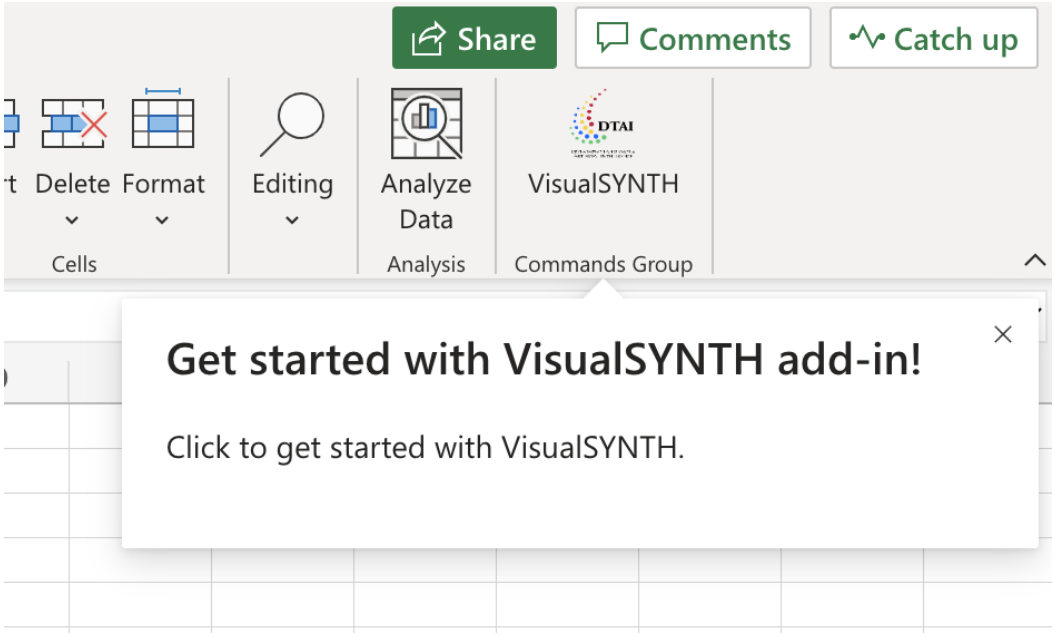
Once the download is completed, one can upload the Add-in in Excel through the online services from Microsoft Office. This can be done through the Insert Menu by clicking on the Office Add-ins button (see Fig. 1).

Figure 1. Uploading a new Office Addin in Microsoft Excel.



Once the upload is done, the Add-in will appear under the Home Menu (see Fig. 2). By clicking on it, the Add-in will appear, and from there, one can interact and use the functionality offered by VisualSYNTH.

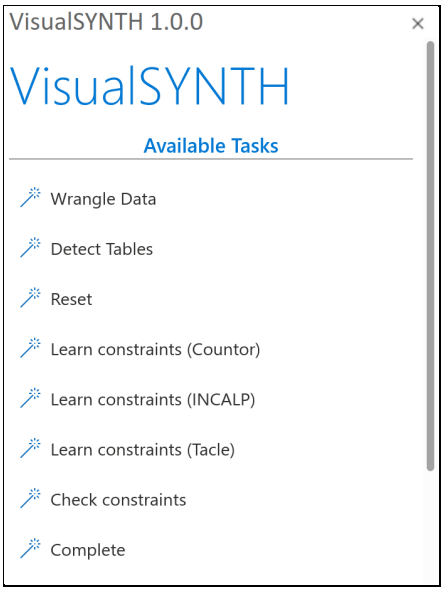
Figure 2. VisualSYNTH Add-in under the Home Menu in Microsoft Excel.



Interacting with VisualSynth

Sidebar

Interacting with the tool mostly requires selecting one of the available tasks in the sidebar on the right.



Initially, the only available task is *Detect Tables*. All tasks operate on tables, so making sure that they are properly detected is a prerequisite.

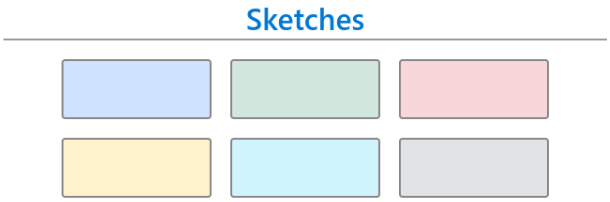
Depending on the internal state of the tool, different tasks become available. At any point in time, the user can *Reset* the state of all current data science tasks. The *Reset* task will remove all learned models and other helpers (such as tables) that are needed for the tool to provide data science tasks.

Sketches

Sometimes, VisualSynth cannot figure out how to execute a certain task without additional supervision from the user. *Sketches* address this problem by providing additional input to the tool. Through simply selecting the target cells and clicking one of the colors in the tool, the user can instruct VisualSynth on the cells relevant for the desired task. How this information is used depends on the task at hand and will be explained in their respective sections. At any point, the user can revert the sketches using the *Clear* button.

Quick tips.

- Multiple cells can be selected at once by holding CTRL.
- Sketch colors are independent from the colors you see around detected tables.
- Colors do not have specific meaning. Rather, a configuration of the same or different colors provides the information to the task at hand.



Formatting data

Tables in data science should follow an *attribute-value* format, where every row describes a specific datapoint and every column describes one attribute or feature of this datapoint.

Attribute 1	Attribute 2	Attribute 3
TRUE	0.2	3
FALSE	0.7	7
TRUE	9.4	666
TRUE	1.2	42

Many real world spreadsheets are not yet properly formatted this way. After detecting tables, but before proceeding with any data science task, it is necessary to convert the data to an attribute-value format. This process is called **data wrangling**.

To run the wrangling task, press *Wrangle Data*. In many cases, VisualSynth is able to

automatically detect the right format without supervision.

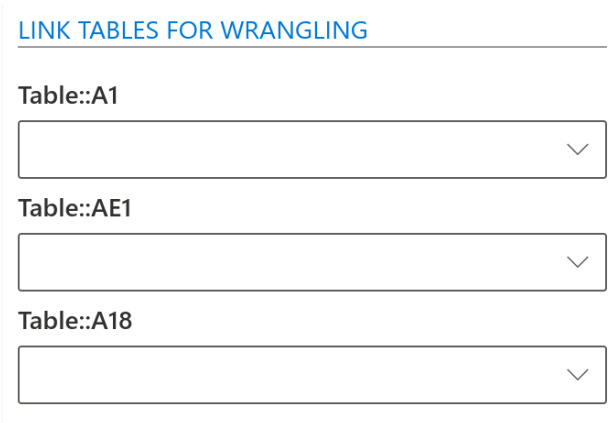
Sometimes, however, it is hard to automatically figure out what exactly we want. For example, consider the two views on the same dataset pictured below (Right picture is incorrect). Both can be considered attribute-value, as the sales number in a specific month can be considered a feature as well. Depending on the task at hand, either format can be correct.

Type	Country	June	July	August	Profit
Banana	BE	170	690	520	YES
Banana	DE	610	640	320	NO
Banana	FR	250	650	630	YES
Chocolate	BE	560	320	140	YES
Chocolate	FR	430	350	300	YES
Chocolate	NL	210	280	270	NO
Speculaas	BE	300	270	290	NO
Vanilla	BE	610	190	670	YES

Type	Country	June	July	Profit
Banana	BE	June	170	YES
Banana	BE	July	690	YES
Banana	BE	August	520	YES
Banana	DE	June	610	NO
Banana	DE	July	640	NO
Banana	DE	August	320	NO
		...		
Vanilla	BE	June	610	YES
Vanilla	BE	July	190	YES
Vanilla	BE	August	670	YES

In such cases, we can add some form of supervision by using sketches. By default, the left format is returned—there is no specific reason to unpivot the month columns.[1] More specifically, we can mark some of the cells that should end up in the same datapoint, and thus describe the properties of one datapoint.

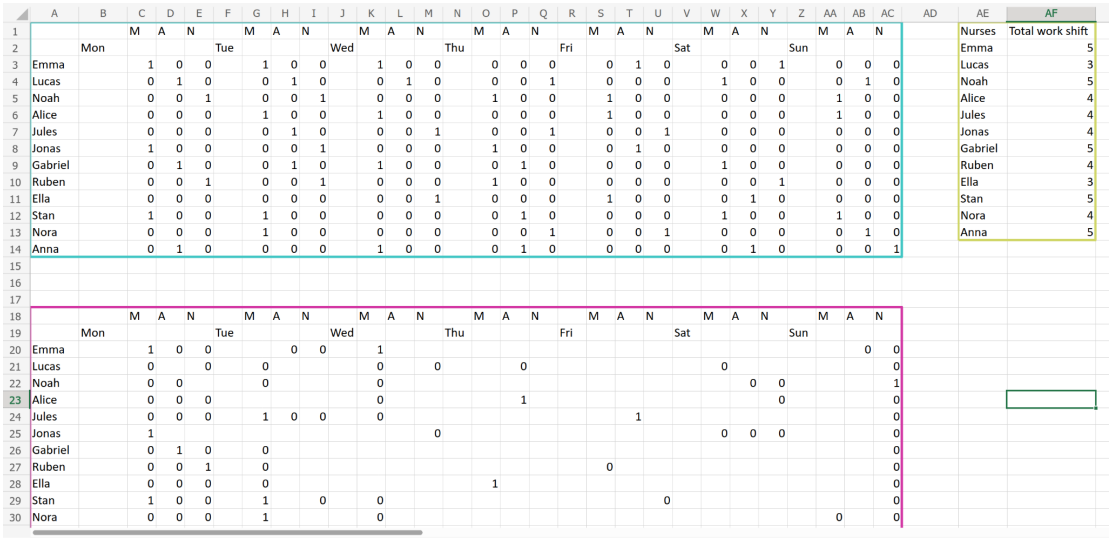
Additionally, different tables in a spreadsheet could share the same format. After detecting the tables, it is possible to *link* them.



[1] In smart

For instance, running *Data Wrangling* after linking tables B and C to table A will try to infer a conversion procedure from table A and then apply it to A, B and C.

Consider, for example, the following spreadsheet containing the shifts of nurses of a hospital. Schedules display for every nurse whether they work on a specific day and shift. The schedule itself then contains for every combination of nurse, day and shift a 1 (the nurse works on that day and shift) or 0 (the nurse does not work on that day and shift). Crucially, notice that the tables are not in attribute-value format and thus must be wrangled. There are 3 tables in this example: the two tables on the left contain a complete and a partial schedule while, on the right, a table indicates the total number of shifts for each nurse. VisualSynth should be able to automatically detect them, as depicted below.

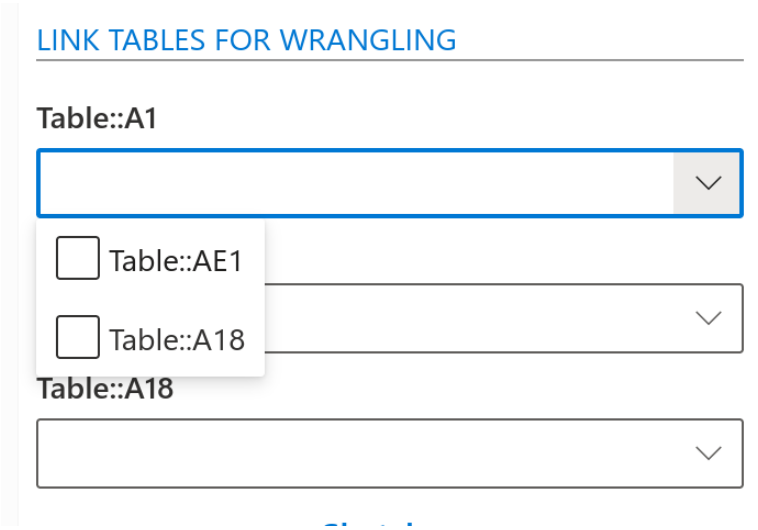


	Mon			Tue			Wed			Thu			Fri			Sat			Sun		
	M	A	N	M	A	N	M	A	N	M	A	N	M	A	N	M	A	N	M	A	N
Emma	1	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0
Lucas	0	1	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0
Noah	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0
Alice	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0
Jules	0	0	0	0	1	0	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0
Jonas	1	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0
Gabriel	0	1	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Ruben	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0
Ella	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0
Stan	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0
Nora	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0
Anna	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	1

Nurses	Total work shift
Emma	5
Lucas	3
Noah	5
Alice	4
Jules	4
Jonas	4
Gabriel	5
Ruben	4
Ella	3
Stan	5
Nora	4
Anna	5

	Mon			Tue			Wed			Thu			Fri			Sat			Sun		
	M	A	N	M	A	N	M	A	N	M	A	N	M	A	N	M	A	N	M	A	N
Emma	1	0	0			0	0	1												0	0
Lucas	0		0	0			0		0			0						0			0
Noah	0	0		0			0										0	0			1
Alice	0	0	0				0					1						0			0
Jules	0	0	0	1	0	0	0								1						0
Jonas	1								0							0	0	0			0
Gabriel	0	1	0	0																	0
Ruben	0	0	1	0											0						0
Ella	0	0	0	0									1								0
Stan	1	0	0	1	0	0	0									0					0
Nora	0	0	0	1			0												0		0

Before executing the data wrangling procedure, the incomplete schedule (Table::A18) should be linked to the complete one (Table::A1), as they clearly share the same format.



In this setting, VisualSynth cannot automatically infer the right format for the data and needs some supervision from the user. Using sketches to highlight the fields that should end up in the same row (in Table::A1) provides sufficient information to VisualSynth to infer the correct conversion procedure.

	A	B	C	D	E	F	G	H	I	J
1			M	A	N		M	A	N	
2		Mon				Tue				We
3	Emma		1	0	0		1	0	0	
4	Lucas		0	1	0		0	1	0	
5	Noah		0	0	1		0	0	1	
6	Alice		0	0	0		1	0	0	
7	Jules		0	0	0		0	1	0	
8	Jonas		1	0	0		0	0	1	
9	Gabriel		0	1	0		0	1	0	

Subsequently running *Wrangle Data* will create a new spreadsheet where the schedules are flattened in an attribute-value format.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Emma	Mon	M	1		Emma	Mon	M	0		Nurses	Total work shift	
2	Emma	Mon	A	0		Emma	Mon	A	1		Emma	3	
3	Emma	Mon	N	0		Emma	Mon	N	0		Lucas	3	
4	Lucas	Mon	M	0		Lucas	Mon	M	0		Noah	5	
5	Lucas	Mon	A	1		Lucas	Mon	A	0		Alice	4	
6	Lucas	Mon	N	0		Lucas	Mon	N	0		Jules	4	
7	Noah	Mon	M	0		Noah	Mon	M	0		Jonas	4	
8	Noah	Mon	A	0		Noah	Mon	A	0		Gabriel	5	
9	Noah	Mon	N	1		Noah	Mon	N	0		Ruben	4	
10	Alice	Mon	M	0		Alice	Mon	M	0		Ella	3	
11	Alice	Mon	A	0		Alice	Mon	A	0		Stan	5	
12	Alice	Mon	N	0		Alice	Mon	N	0		Nora	4	
13	Jules	Mon	M	0		Jules	Mon	M	0		Anna	5	
14	Jules	Mon	A	0		Jules	Mon	A	0				
15	Jules	Mon	N	0		Jules	Mon	N	0				
16	Jonas	Mon	M	1		Jonas	Mon	M	1				
17	Jonas	Mon	A	0		Jonas	Mon	A	0				
18	Jonas	Mon	N	0		Jonas	Mon	N	0				
19	Gabriel	Mon	M	0		Gabriel	Mon	M	0				
20	Gabriel	Mon	A	1		Gabriel	Mon	A	1				
21	Gabriel	Mon	N	0		Gabriel	Mon	N	0				
22	Ruben	Mon	M	0		Ruben	Mon	M	0				
23	Ruben	Mon	A	0		Ruben	Mon	A	0				
24	Ruben	Mon	N	1		Ruben	Mon	N	1				
25	Ella	Mon	M	0		Ella	Mon	M	0				

Cleaning data

Detecting Errors

Cleaning data is a multi-step process. The first step is to remove erroneous data from the dataset. *Anomaly detection* highlights abnormal data points which you can then investigate to find errors and correct them or remove them from the data.

Selecting Relevant Data

Often, only a subset of the data is of interest to a specific task. The *smart select* task makes it easy to select a subset of your data. Using sketches -- see the section on formatting -- you can give examples of rows you do and do not want to include. Simply sketch the rows in different colors. The smart select task then assigns a color to all other rows, finding criteria that can be used to separate the rows of different colors.[2]

Below is an example of a few selected rows (the sketch) and the result of smart filling the remainder of the table.

[2] Soon, functionality will be included to split a specific color off to a new table.

Banana	BE	170	690	520	1380	YES
Banana	DE	610	640	320	1570	NO
Banana	DE	250	650	630	1530	YES
Chocolate	BE	560	320	140	1020	YES
Chocolate	FR	430	350	300	1080	YES
Chocolate	NL	210	280	270	760	NO
Speculaas	BE	300	270	290	860	NO
Speculaas	NL	290	250	210	750	NO
Vanilla	BE	610	190	670	1470	YES

Banana	BE	170	690	520	1380	YES
Banana	DE	610	640	320	1570	NO
Banana	DE	250	650	630	1530	YES
Chocolate	BE	560	320	140	1020	YES
Chocolate	FR	430	350	300	1080	YES
Chocolate	NL	210	280	270	760	NO
Speculaas	BE	300	270	290	860	NO
Speculaas	NL	290	250	210	750	NO
Vanilla	BE	610	190	670	1470	YES

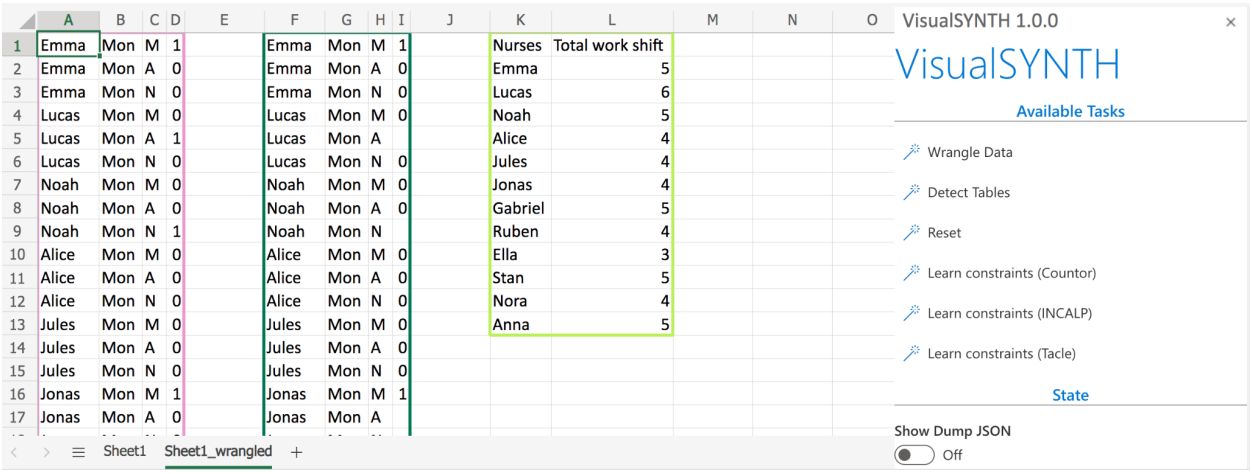
Constraints

Constraints are pervasive in Operations Research (OR) applications such as scheduling and planning. Consider for instance the case of nurse scheduling. Hospitals usually generate a weekly schedule for their nurses based on constraints like “a nurse can work at most 5 days every week”. As the number of nurses and the complexity of the constraints increases, generating the schedule manually becomes impractical. Hospitals can use optimization solvers (e.g. Gurobi) to produce the schedules automatically, but these solvers require modeling the constraints into an OR model, which itself is a complicated task. Hiring a domain and modeling expert to manually design a model can be expensive and time consuming. Our tool can help in automating this process.

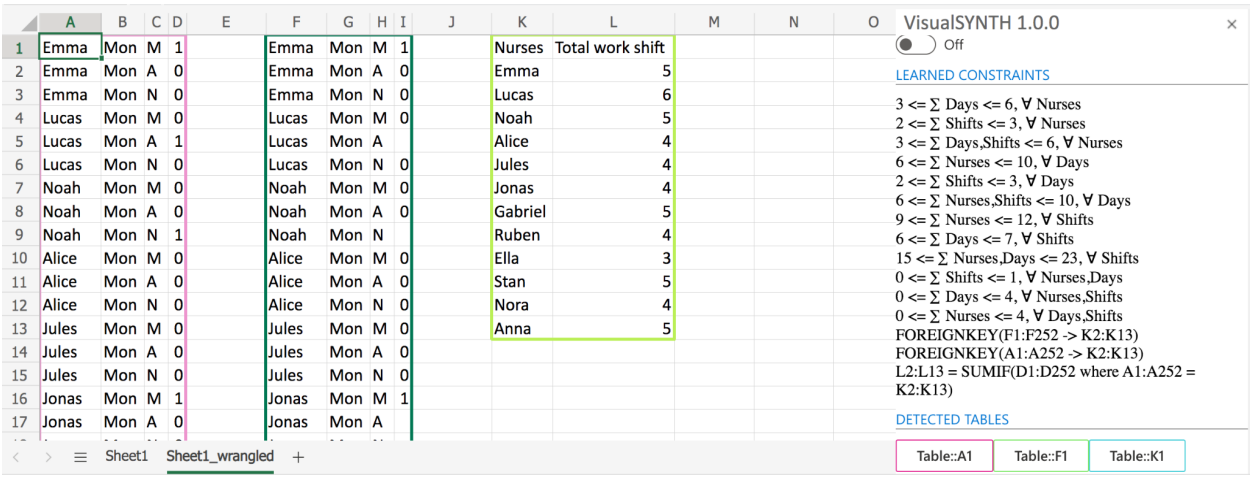
COUNTOR

COUNT-OR can automatically learn constraints from past schedules and then encode it into a Gurobi model. This model then can be used to generate new schedules. To demonstrate this, we will work with nurse schedules, which is provided as a table in an excel sheet. Using COUNT-OR we will first learn all the constraints and then use these constraints to autofill partial schedules in the sheet. These partial schedules can be seen as nurse preferences which have already been put in the schedule, and then COUNT-OR helps in scheduling nurses in the empty shifts, making sure that the learned constraints are satisfied. To do this, we first wrangle the data using sketch, as explained previously. Afterwards, we take the following steps on the wrangled data:

1. Detect tables: We start by detecting all the tables in the wrangled data



2. Learn constraints: We then learn the constraints using CountOR and TaCLe



3. Autocomplete missing values: Finally, we use the learned constraints to complete the partial schedule.

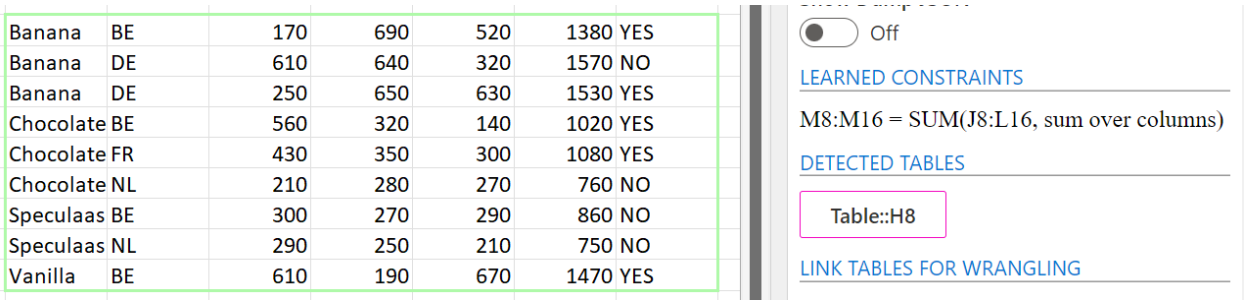
	A	B	C	D	E	F	G	H	I	J	K	L
125	Emma	Mon	M	1		Emma	Mon	M	0		Nurses	Total work shift
126	Emma	Mon	A	0		Emma	Mon	A	0		Emma	5
127	Emma	Mon	N	0		Emma	Mon	N	0		Lucas	6
128	Lucas	Mon	M	0		Lucas	Mon	M	0		Noah	5
129	Lucas	Mon	A	1		Lucas	Mon	A	0		Alice	4
130	Lucas	Mon	N	0		Lucas	Mon	N	0		Jules	4
131	Noah	Mon	M	0		Noah	Mon	M	0		Jonas	4
132	Noah	Mon	A	0		Noah	Mon	A	0		Gabriel	5
133	Noah	Mon	N	1		Noah	Mon	N	0		Ruben	4
134	Alice	Mon	M	0		Alice	Mon	M	0		Ella	3
135	Alice	Mon	A	0		Alice	Mon	A	0		Stan	5
136	Alice	Mon	N	0		Alice	Mon	N	0		Nora	4
137	Jules	Mon	M	0		Jules	Mon	M	0		Anna	5
138	Jules	Mon	A	0		Jules	Mon	A	0			

4. Update: Finally, if the user is not happy with the generated schedule, they can modify some values manually and call the autocompletion again.

TaCLe

While scheduling applications often rely on constraints to find schedules, simple constraints occur across a wide range of applications. In spreadsheets we often see formulas, such as SUM or SUMIF that can be modeled as constraints between the input cells and the output cells. In cases where these formulas are not explicitly present in the Excel sheet, for example, when loading a CSV file, the TaCLe constraint learner can discover such constraints in the data and translate them into the same format as other constraints used by SYNTH. These spreadsheet constraints can then be used as part of multi-cell predictions, keeping tables in sync or to impose additional rules on schedules.

In the example below, TaCLe has identified the fact that the sixth column is the sum of the previous columns.



Predictions

Using predictive models and constraints it is possible to predict values in your data that are missing or not yet known. To predict empty cells, just use the *Smart fill* task from the sidebar. Smart fill then automatically encodes the data in a machine understandable format, learns suitable predictive models and incorporates any constraints that have previously been learned. Then it predicts the missing cells, making sure predictions adhere to the learned constraints.

The VisualSYNTH tool displays a background color for every prediction that shows the degree of confidence of the predictive model that produced the prediction. This can give you an indication of how trustworthy a prediction is (green is better, red is worse). But be aware that in some cases predictive models can be misled to be overconfident. Predictions made using constraints are guaranteed to fulfill the constraints and are therefore always indicated as trustworthy.