

HE Plant Digital Strategy

*Empowering Self-Service Analytics
with Seeq and GenAI*



Ajish Varghese

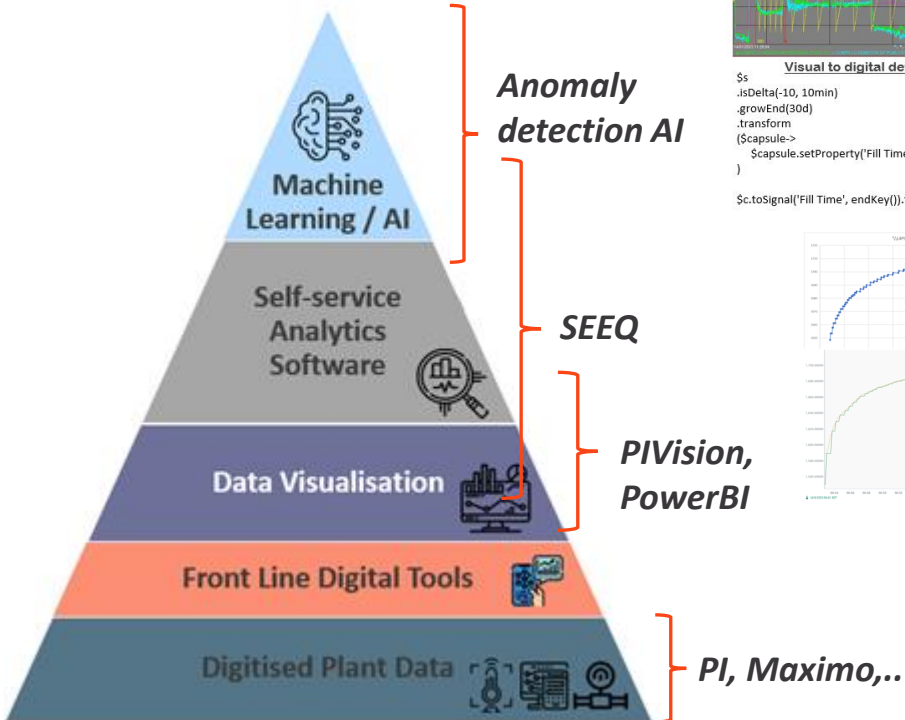


Plant Digital Strategy & SEEQ

- Seeq - cloud-based solution that enables users to get more value and faster time to insight from timeseries data (from equipment, process, well, drilling, production etc.)
 - 2022: Proof of Value (PoV) trial with extracted data
 - 2023: Extended PoV with live connection to PI
 - 2024 (ongoing): UK implementation 'Q4



HE Plant Digital Tools

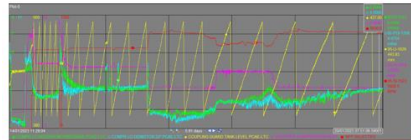


Anomaly detection AI

SEEQ

PI Vision, PowerBI

PI, Maximo,...



```

Visual to digital determination of fill times
$S
JsDelta(-10, 10min)
.growEnd(30d)
.transform
($capsule->
  $capsule.setProperty('Fill Time',$capsule.duration().convertUnits('h'))
)
Sc.toSignal('Fill Time',endKey()).toStep()
    
```

Digitise

The screenshot shows a software interface with a search bar, a list of variables, and a formula editor. The formula editor contains the following code:

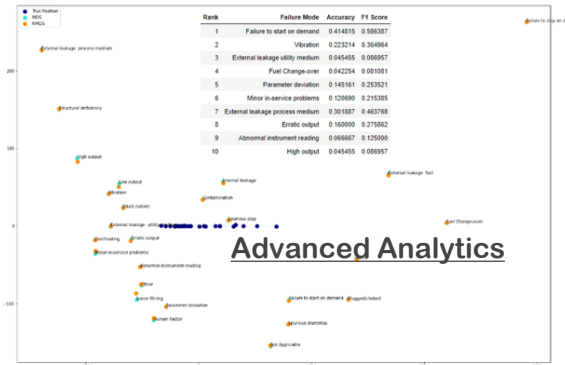
```

Reliability_Utl11_duplicatetransients()
duplicatetransients will remove consecutive duplicate values that
exist within a set-off period from the input signal to give a smoothly
interpolated signal during a transient period. The additional
arguments are the maximum period between samples for
deduplication and the maximum interpolation for the resulting signal.
Variables
Reliability_Utl11_duplicatetransients(%_ts, maximum:
  %_ts, %_signal, %_interpolation:
Parameters
maximum: Scalar
%_ts: %_Signal
%_signal: %_Signal
Save As: Reliability_Utl11_duplicatetransients.vi
    
```

Cleanse

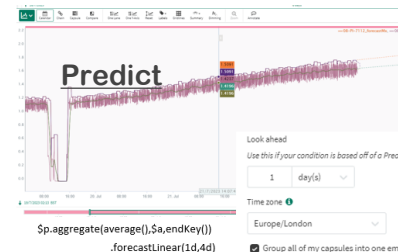


Model

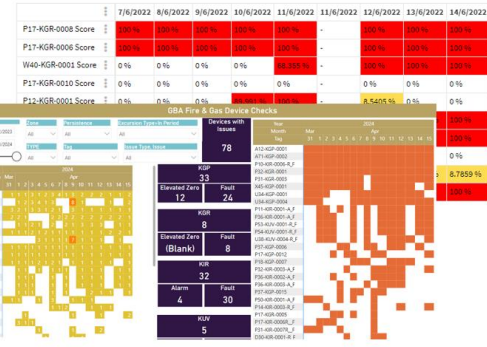


Advanced Analytics

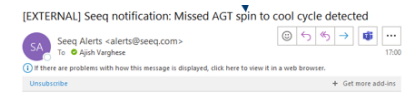
^GenAI (native / external)



Predict



Visualisation & Integration



Advanced alerts

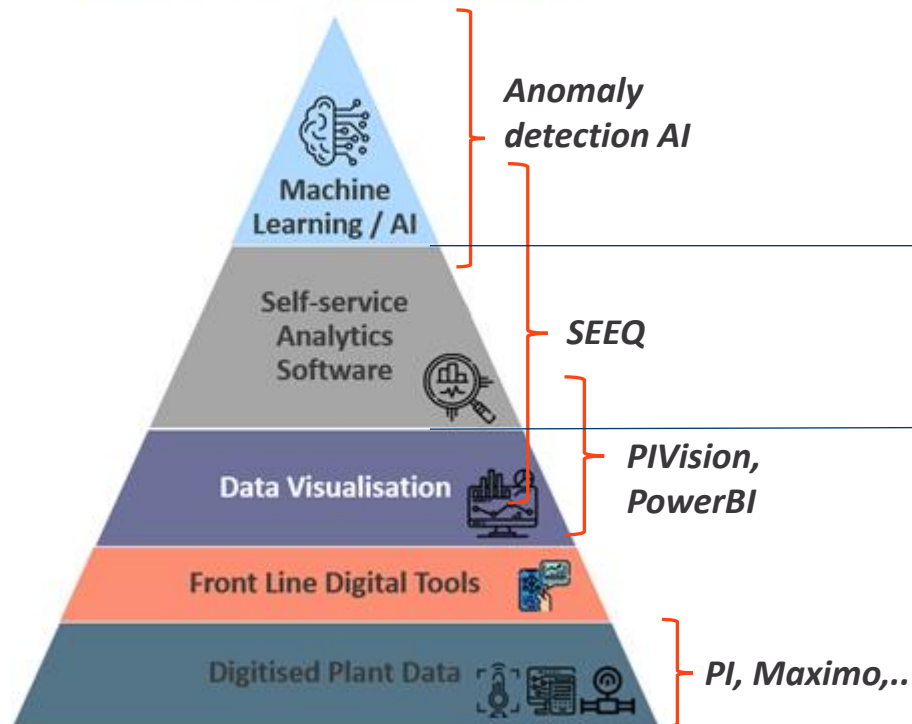
```

Please start spin to cool cycle if possible.
See more information in Seeq: LSPC-1
New capsules detected for NoSPC_notification:
Start AGT
Jan 30, 2024 4:59 PM AGT3
Use this if your condition is based off of a Predict
combineWith($x, $y, $z)
    
```

Plant Digital Strategy - Copilot

- GenAI/Copilot accelerates productivity at all levels – operational efficiency and faster innovation

HE Plant Digital Tools



- L1: ML/AI support – implement pagination to obtain anomaly detection model learn ranges (PowerBI/PowerQuery)
- L2: Self-service Analytics support – step by step instructions for analysis and calculations. e.g. tank fill times in Seeq
- L3+: Visualisation & Data analysis support
 - Data cleaning for PI (Excel)
 - Calculations / measures (PowerBI/DAX)
 - SQL statements
 - Exploratory Data Analysis and more..

Copilot – L1 support

PowerQuery script update:

- Starting script fetches all records in one call
- Need to split to batches of 50

Copilot results:

- First version – recursive handling, not optimal
- Second iteration – loop based, no error handling
- Final version – after error handling iterations

```
let
Source = Json.Document(....., [Headers = [Authorization = apiauth]]),
#"Converted to Table1" = Table.FromList(Source, Splitter.SplitByNothing(), null, null, ExtraValues.Error),
#"Expanded Column2" = Table.ExpandRecordColumn(#"Converted to Table1", "Column1", {"id", "spec"}, {"id", "spec"}),
#"Expanded spec" = Table.ExpandRecordColumn(#"Expanded Column2", "spec", {"timeRanges"}, {"timeRanges"}),
#"Expanded timeRanges" = Table.ExpandListColumn(#"Expanded spec", "timeRanges"),
#"Expanded timeRanges1" = Table.ExpandRecordColumn(#"Expanded timeRanges", "timeRanges", {"startTime", "endTime"}, {"startTime", "endTime"}),
#"Added Custom" = Table.AddColumn(#"Expanded timeRanges1", "Period", each {startTime, endTime, type text}, type text),
#"Grouped Rows" = Table.Group(#"Added Custom", {"id"}, {"Periods", each _ type table [id=text, startTime=text, endTime=text, Period=text]}),
#"Aggregated Periods" = Table.AggregateTableColumn(#"Grouped Rows", "Periods", {"Period", each Text.Combine(_, " ")}),
in
#"Aggregated Periods"
```

```
let
// Function to fetch data with pagination
FetchData = (offset as number, limit as number) as table =>
let
Source = Json.Document(....., offset=Text.From(offset), limit=Text.From(limit)),
Headers = [Authorization = apiauth]],
ConvertedToTable = Table.FromList(Source, Splitter.SplitByNothing(), null, null, ExtraValues.Error),
ExpandedColumn = Table.ExpandRecordColumn(ConvertedToTable, "Column1", {"id", "spec"}, {"id", "spec"}),
ExpandedSpec1 = Table.ExpandRecordColumn(ExpandedColumn, "spec", {"timeRanges"}, {"timeRanges"}),
ExpandedTimeRanges = Table.ExpandListColumn(ExpandedSpec1, "timeRanges"),
ExpandedTimeRanges1 = Table.ExpandRecordColumn(ExpandedTimeRanges, "timeRanges", {"startTime", "endTime"}, {"startTime", "endTime"}),
AddedCustom = Table.AddColumn(ExpandedTimeRanges1, "Period", each {startTime, endTime, type text}, type text),
in
AddedCustom

// Function to fetch data with pagination
FetchData = (offset as number, limit as number) as list =>
let
Source = Json.Document(....., offset=Text.From(offset), limit=Text.From(limit)),
Headers = [Authorization = apiauth]],
ConvertedToTable = Table.FromList(Source, Splitter.SplitByNothing(), null, null, ExtraValues.Error),
ExpandedColumn = Table.ExpandRecordColumn(ConvertedToTable, "Column1", {"id", "spec"}, {"id", "spec"}),
ExpandedSpec = Table.ExpandRecordColumn(ExpandedColumn, "spec", {"timeRanges"}, {"timeRanges"}),
ExpandedTimeRanges = Table.ExpandListColumn(ExpandedSpec, "timeRanges"),
ExpandedTimeRanges1 = Table.ExpandRecordColumn(ExpandedTimeRanges, "timeRanges", {"startTime", "endTime"}, {"startTime", "endTime"}),
AddedCustom = Table.AddColumn(ExpandedTimeRanges1, "Period", each {startTime, endTime, type text}, type text),
in
AddedCustom

// Function to fetch data with pagination
FetchData = (offset as number, limit as number) as list =>
let
Source = Json.Document(....., offset=Text.From(offset), limit=Text.From(limit)), Headers = [Authorization = apiauth]]
in
Source,

// Initialize variables
offset = 0,
limit = 50,
allData = FetchData(offset, limit),
currentBatch = allData,

// Loop to fetch all data
FetchLoop = List.Generate(
() => [offset = offset, data = currentBatch],
each List.Count([data]) > 0,
each [offset = [offset] + limit, data = FetchData([offset], limit)],
each [data]
),

// Combine all batches into one list
CombinedList = List.Combine(FetchLoop),

// Convert combined list to table
CombinedTable = Table.FromList(CombinedList, Splitter.SplitByNothing(), null, null, ExtraValues.Error),

// Expand the records
ExpandedColumn = Table.ExpandRecordColumn(CombinedTable, "Column1", {"id", "spec"}, {"id", "spec"}),
ExpandedSpec = Table.ExpandRecordColumn(ExpandedColumn, "spec", {"timeRanges"}, {"timeRanges"}),
ExpandedTimeRanges = Table.ExpandListColumn(ExpandedSpec, "timeRanges"),
ExpandedTimeRanges1 = Table.ExpandRecordColumn(ExpandedTimeRanges, "timeRanges", {"startTime", "endTime"}, {"startTime", "endTime"}),
AddedCustom = Table.AddColumn(ExpandedTimeRanges1, "Period", each {startTime, endTime, type text}, type text),

// Group and aggregate periods
GroupedRows = Table.Group(AddedCustom, {"id"}, {"Periods", each _ type table [id=text, startTime=text, endTime=text, Period=text]}),
AggregatedPeriods = Table.AggregateTableColumn(GroupedRows, "Periods", {"Period", each Text.Combine(_, " ")}),
in
AggregatedPeriods
```

Copilot – L2 support

■ Seeq analysis – determine tank fill intervals

– Prompt

You
I have a time series signal in Seeq software that shows the level in a tank that fills up relatively steadily over time, reaches a high level and empties, and fills up again and so on. I want to calculate the filling time as another signal that shows the value of previous completed filling. Explain how I can achieve this in Seeq alongwith the relevant calculations.

– Returns steps, calculations, and doing it in Seeq

– Address noise issue

You
using the naive derivative does not work well due to measurement noise in the signal. use a more robust approach.

Copilot
To handle measurement noise in your signal, you can use a more robust approach to calculate the derivative. One effective method is to use a **smoothing filter** before calculating the derivative. Here's how you can do it in Seeq:

Step 1: Apply a Smoothing Filter
Use a smoothing filter like a moving average or a low-pass filter to reduce noise in your signal.

```
Python
$level = $yourTankLevelSignal
$$smoothed_level = $level.aggregate(average(), $level.timePeriod(1min), startKey())
```

– Fixes

- Data driven parameter values
- Function parameter issues
- Function choices

– SME result:

```
$s
.isDelta(-10, 10min)
.growEnd(30d)
.transform
($capsule->
  $capsule.setProperty('Fill Time',$capsule
)
$.toSignal('Fill Time', endKey()).toStep()
```

Copilot
To calculate the filling time in Seeq and create a signal that shows the value of the previous completed filling you can follow these steps:

- 1. Identify the Filling and Emptying Events:**
 - Use the **Value Search** tool to create conditions that identify when the tank is filling and when it is emptying. For example, you can create a condition for when the level is increasing (filling) and another for when it is decreasing (emptying).
- 2. Create Capsules for Each Filling Event:**
 - Use the **Capsule View** to visualize these conditions. You can create capsules that represent each filling event by combining the conditions for the start and end of the filling process.
- 3. Calculate the Duration of Each Filling Event:**
 - Use the **Signal from Condition** tool to calculate the duration of each filling event. This tool allows you to create a new signal that represents the duration of each capsule (filling event).
- 4. Create a Signal for the Previous Completed Filling:**
 - Use the **Formula** tool to shift the calculated duration signal by one event. This will create a new signal that shows the duration of the previous filling event.

Step 1: Identify the Filling and Emptying Events

```
Python
$filling = $level.isDelta(10, 10min)
$emptying = $level.isDelta(-10, 10min)
```

Step 2: Create Capsules for Each Filling Event

```
Python
$fillingCapsules = $filling.and($emptying.inverse())
```

Step 3: Calculate the Duration of Each Filling Event

```
Python
$fillingDuration = $fillingCapsules.signalFromCondition()
```

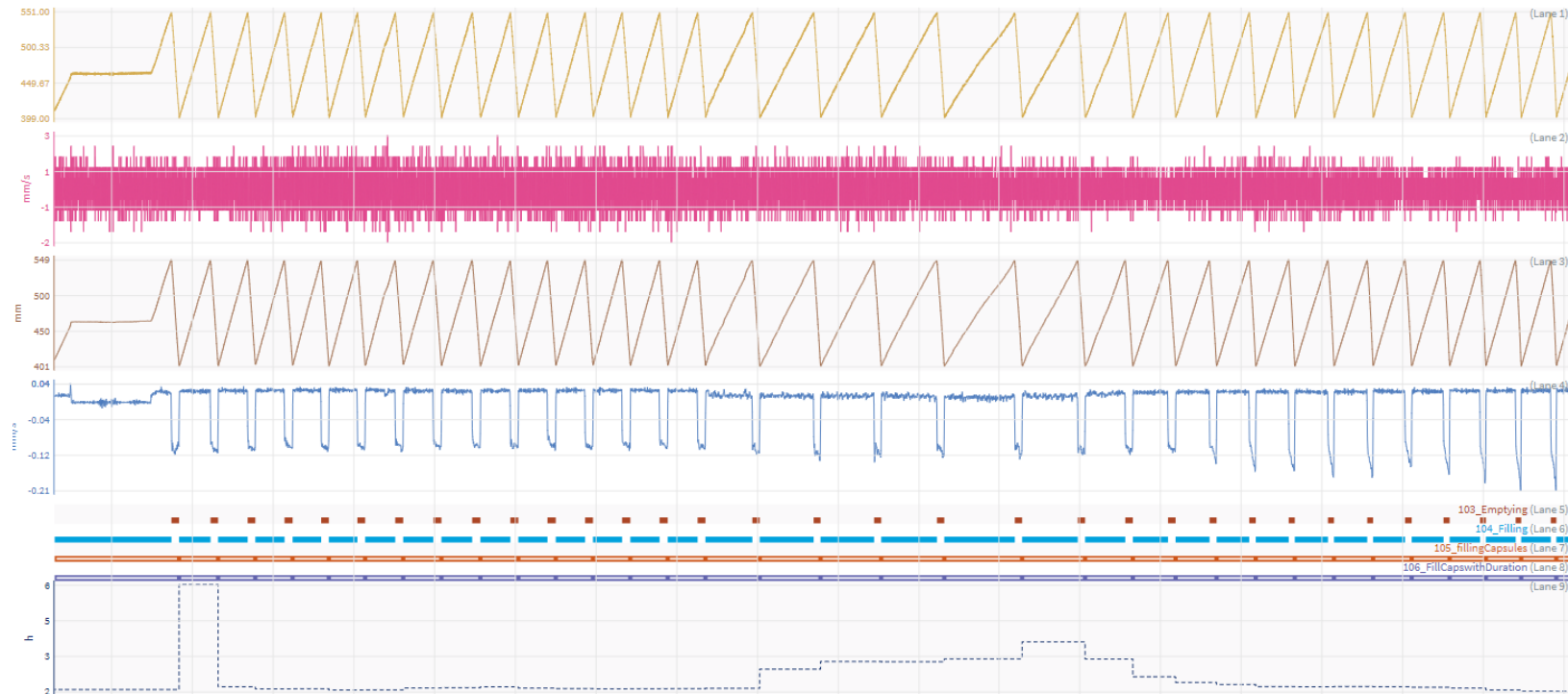
Step 4: Create a Signal for the Previous Completed Filling

```
Python
$previousFillingDuration = $fillingDuration.shift(1, $fillingDuration.startKey())
```

Putting It All Together

- 1. Create the conditions for filling and emptying:**
 - Use the **Value Search** tool to create conditions for when the tank is filling and emptying.
- 2. Create capsules for each filling event:**
 - Use the **Capsule View** to visualize and combine these conditions.
- 3. Calculate the duration of each filling event:**
 - Use the **Signal from Condition** tool to create a signal representing the duration of each filling event.
- 4. Shift the duration signal:**
 - Use the **Formula** tool to shift the duration signal by one event to get the previous filling duration.

By following these steps, you can create a signal in Seeq that shows the duration of the previous completed filling event. If you need more detailed guidance or run into any issues, feel free to ask!



Copilot – L3+ support

DAX measure

- Needed update to allow different trend calculation and filtering behaviour

- Prompt

You

write a DAX measure to calculate:

- for each distinct value of column Y the number of rows in a PowerBI table R, divide this rowcount by the value of Y, using 5 as the default if the value does not exist or is erroneous
- return a twelfth of their rounded up total

- Results

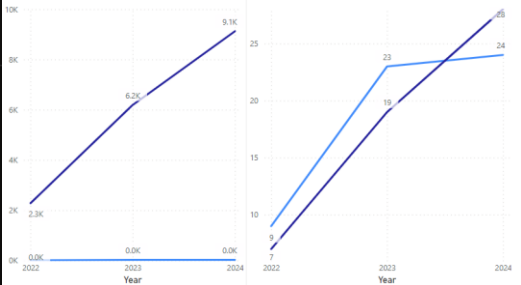
This measure works as follows:

1. `DefaultY` sets the default value to 5.
2. `RowCounts` calculates the number of rows for each distinct value of `Y`.
3. `DividedCounts` divides each row count by the corresponding value of `Y`, using the default value of 5 if `Y` is blank or not a number.
4. `TotalDividedCount` sums these divided counts.
5. Finally, `CEILING(TotalDividedCount / 12, 1)` calculates a twelfth of the total and rounds it up to the next whole number.

Feel free to ask if you need any further adjustments or explanations!

Copilot

Here's a DAX measure that calculates the number of rows in table `R` for each distinct value of column `Y`, divides this row count by the value of `Y` (using 5 as the default if the value does not exist or is erroneous), and returns a twelfth of their rounded-up total:



```

RoundedRatioPerY =
VAR DefaultY = 5
VAR RowCounts =
    ADDCOLUMNS(
        SUMMARIZE(R, R[Y]),
        "RowCount", COUNTROWS(FILTER(R, R[Y] = EARLIER(R[Y])))
    )
VAR DividedCounts =
    ADDCOLUMNS(
        RowCounts,
        "DividedCount",
        DIVIDE([RowCount],
            IF(ISBLANK(R[Y]) || NOT(ISNUMBER(R[Y])), DefaultY, R[Y])
        )
    )
VAR TotalDividedCount =
    SUMX(DividedCounts, [DividedCount])
RETURN
    CEILING(TotalDividedCount / 12, 1)

```

Copilot – L3+ support

Excel formula:

- Lookup full tag names from instrument ids >>
 - Output as method, optionally generate the result sheet

SQL statement update:

- Update base query returning all data to:
 - Filter to only last calendar month based on a date field
 - Group the results by a set combination of fields
 - Keep all the highest category values only, based on a custom order of a field, aggregating specific columns on the common value and concatenating distinct values on the remaining columns
 - Subsequently, for two sets of descriptive field pairs: concatenate values of the pair when either is empty, otherwise use their distinct value when they're the same otherwise concatenate with a new line
 - Keeping the resulting query readable and efficient
- Result
 - Desired output with some iterations/guidance required on efficiency, function usage and logic

You

Given an excel file with two sheets of interest - 'Sheet1' with a column called 'sstr' and 'Sheet2' with a column called 'tag';
your task is to construct an excel formula for a new result column beside in the 'sstr' column in 'Sheet1' that finds all the values in the 'tag' column of 'Sheet2' sheet that contains the value in the 'sstr' column, and concatenates them with a semi-colon when there are multiple matches and sets 'Not Found' when there are no matches.

Copilot

Sure, I can help with that! You can use a combination of Excel functions to achieve this. Here's a formula you can use in the new result column beside the 'sstr' column in 'Sheet1':

```
=IFERROR(TEXTJOIN(";", TRUE, FILTER(Sheet2!B:B, ISNUMBER(SEARCH(Sheet1!A2, Sheet2!B:B)), "")), "Not Found")
```

Here's a breakdown of what this formula does:

- SEARCH(Sheet1!A2, Sheet2!B:B):** This searches for the value in 'sstr' (Sheet1!A2) within the 'tag' column (Sheet2!B:B). It returns an array of positions where the value is found.
- ISNUMBER(SEARCH(Sheet1!A2, Sheet2!B:B)):** This converts the array of positions into TRUE/FALSE values, where TRUE indicates a match.
- FILTER(Sheet2!B:B, ISNUMBER(SEARCH(Sheet1!A2, Sheet2!B:B)), ""):** This filters the 'tag' column to include only the values that contain the 'sstr' value.
- TEXTJOIN(";", TRUE, ...):** This concatenates the filtered values with a semi-colon.
- IFERROR(..., "Not Found"):** This handles cases where no matches are found, returning "Not Found".

Place this formula in the new result column beside the 'sstr' column in 'Sheet1', starting from the first row of data (e.g., B2 if 'sstr' is in A2).