Here are **5 detailed, real‑world examples** of using **TabPy / SCRIPT\_ functions** in **Tableau Desktop** (with the Superstore-style data) — including **full explanation** of how it works, where to put the script, and sample code. These are “live” calculations: Tableau sends the data to Python (TabPy), that executes code with pandas/numpy, and returns results to be visualized.

**⚙️ Setup Reminder (Prerequisites)**

Before any of these will work, you must:

1. **Install TabPy** in your Python environment:
2. pip install tabpy
3. tabpy

(This runs the TabPy server, listening by default on localhost:9004.)

1. **Connect Tableau Desktop to TabPy**:
   * In Tableau: **Help → Settings and Performance → Manage External Service Connection**
   * Set Service = *TabPy / External API*
   * Host = localhost, Port = 9004
   * Click **Test Connection** (should show success)
2. Use the **SCRIPT\_** functions (SCRIPT\_REAL, SCRIPT\_INT, SCRIPT\_STR, SCRIPT\_BOOL) in **calculated fields** in Tableau. Tableau will send aggregates (based on the view’s level of detail) or slices (depending on table calc settings) to Python. [tableau.github.io+2tableau.github.io+2](https://tableau.github.io/TabPy/docs/TableauConfiguration.html?utm_source=chatgpt.com)
3. After you add a Python-based calculated field to the view, you treat it like a **table calculation**: define **“Compute Using”**, partitioning, addressing, restart settings, etc., so that Tableau decides *which rows to send* to TabPy. [help.tableau.com+1](https://help.tableau.com/current/pro/desktop/en-us/r_connection_manage.htm?utm_source=chatgpt.com)

Below are **5 example scenarios** (with script + explanation) you can try with Superstore-like data (fields: Order Date, Region, Category, Sub‑Category, Sales, Profit, etc.).

**1. Running Total of Sales Over Time (by Region or Category)**

**🎯 Purpose**

Cumulatively accumulate sales over time, e.g. show how sales add up month by month for each region (or category).

**🧮 Python Script via SCRIPT\_REAL**

SCRIPT\_REAL(

"

import numpy as np

# \_arg1 is a list/array of sales values, in the order sent by Tableau

return np.cumsum(\_arg1).tolist()

",

SUM([Sales])

)

**📋 How to Use in Tableau**

* Create this calculated field, call it e.g. PY Running Sales.
* Place date (e.g. Order Date in month granularity) on Columns.
* Place Region (or Category) on Color or Rows.
* Put PY Running Sales on Rows (or as measure).
* Edit the table calculation (via “Compute Using”) → set to compute along Order Date.
* If you want separate running totals per region/category, use **“Restarting Every”** = Region (or Category).

**📘 Explanation**

* Tableau will, for each partition (e.g. each Region), send a slice of the *sales values in time order* as \_arg1.
* Python code uses numpy.cumsum() to compute the cumulative sum; returns a list of the same length.
* Tableau assigns those returned values back to each mark in the visualization.
* Equivalent in Tableau native functions: RUNNING\_SUM([Sales]), but here you offload it to Python.

**2. Day-over-Day Change in Profit (Delta)**

**🎯 Purpose**

Calculate the difference (delta) in profit from the previous date, per partition (e.g. by Sub‑Category).

**🧮 Python Script via SCRIPT\_REAL**

SCRIPT\_REAL(

"

import pandas as pd

s = pd.Series(\_arg1)

# diff() gives difference from prior; fill initial with 0

return s.diff().fillna(0).tolist()

",

SUM([Profit])

)

**📋 How to Use in Tableau**

* Create calculated field, e.g. PY Profit Change.
* Use Order Date on Columns, Sub‑Category or Region for partitioning.
* Use PY Profit Change on Rows or as another measure.
* Edit table calculation: “Compute Using” = Order Date.
* Use “Restarting Every” = Sub‑Category (if you want separate for each sub‑category).

**📘 Explanation**

* Python receives a sequence of profit aggregates for each date in the partition.
* Using pandas.Series.diff(), it computes current − previous profit.
* fillna(0) ensures first row has value 0 (or you could choose NaN).
* You now can see day-to-day profit changes visually.

**3. 7-Day Moving Average of Sales**

**🎯 Purpose**

Smooth fluctuations by computing a rolling average (e.g. over past 7 days) of sales.

**🧮 Python Script via SCRIPT\_REAL**

SCRIPT\_REAL(

"

import pandas as pd

s = pd.Series(\_arg1)

# rolling window of 7 rows; at start, smaller window allowed

mov = s.rolling(window=7, min\_periods=1).mean()

return mov.tolist()

",

SUM([Sales])

)

**📋 How to Use in Tableau**

* Create a field PY 7‑Day MA Sales.
* Put Order Date (day-level) on Columns.
* Use Region or Category to partition.
* Use PY 7‑Day MA Sales in Rows (or dual-axis with raw sales).
* Set “Compute Using” = Order Date.

**📘 Explanation**

* Tableau sends sales values in order.
* Python uses pd.Series.rolling(...).mean() to compute the moving average.
* min\_periods=1 ensures early dates still get a value.
* The result gives a smoothed trend line for sales.

**4. Z‑Score Normalization of Sales (Outlier Detection)**

**🎯 Purpose**

Standardize sales values within each partition so you can flag unusually high or low sales (e.g. > 2 standard deviations).

**🧮 Python Script via SCRIPT\_REAL**

SCRIPT\_REAL(

"

import numpy as np

x = np.array(\_arg1, dtype=float)

# compute z = (x − mean) / std, handling zero std

m = x.mean()

std = x.std(ddof=0)

if std == 0:

# if all values equal, return zeros

z = np.zeros\_like(x)

else:

z = (x - m) / std

return z.tolist()

",

SUM([Sales])

)

**📋 How to Use in Tableau**

* Create PY Sales Z‑Score.
* Put Category or Sub‑Category partitioning.
* Use PY Sales Z‑Score on color (e.g. divergent color scale).
* Compute Using → Category or desired dimension.
* Optionally filter to alerts, e.g. ABS([PY Sales Z‑Score]) > 2.

**📘 Explanation**

* For each partition, Python calculates mean and standard deviation.
* It standardizes each sales value.
* You can use it to highlight or filter out extreme values (outliers).

**5. Percent Contribution of Sub‑Category to Category (Within Window)**

**🎯 Purpose**

For a given date (or time grain), show what percentage each sub‑category contributes to its category’s total sales.

**🧮 Python Script via SCRIPT\_REAL**

SCRIPT\_REAL(

"

import numpy as np

x = np.array(\_arg1, dtype=float)

total = x.sum()

# avoid division by zero

if total == 0:

return [0] \* len(x)

pct = (x / total) \* 100

return pct.tolist()

",

SUM([Sales])

)

**📋 How to Use in Tableau**

* Create field PY % of Category Sales.
* Put Category on Rows, Sub‑Category on Columns.
* Use PY % of Category Sales as text/labels or color.
* In table calc dialog, set **Compute Using** = Sub‑Category.
* Use **Restarting Every** = Category (so each category is a separate partition).

**📘 Explanation**

* Tableau sends the sales of all sub‑categories (for that date or partition).
* Python divides each sub‑category sales by the sum of that partition and converts to percentage.
* Each cell shows how much contribution that sub‑category makes within its category.

**🧩 More Notes & Tips (Based on Tableau + TabPy Behavior)**

* **SCRIPT\_ functions are table calculations** — you must control how Tableau slices data (compute using, partitioning). [help.tableau.com+1](https://help.tableau.com/current/pro/desktop/en-us/r_connection_manage.htm?utm_source=chatgpt.com)
* **Aggregation before passing** — you usually wrap measures in aggregation (e.g. SUM([Sales])) before passing to Python. Tableau sends aggregated data per mark or per partition, depending on view. [tableau.github.io+1](https://tableau.github.io/TabPy/docs/TableauConfiguration.html?utm_source=chatgpt.com)
* **Return length must match input length** — Python code should return a list (or array converted to list) with same number of rows as was passed. Otherwise, errors. [tableau.github.io+1](https://tableau.github.io/TabPy/docs/TableauConfiguration.html?utm_source=chatgpt.com)
* **Performance** — Because each partition triggers a Python call, be cautious with large partitions or too many partitions. Minimize number of calls by using “Addressing / Partitioning” smartly. [help.tableau.com](https://help.tableau.com/current/pro/desktop/en-us/r_connection_manage.htm?utm_source=chatgpt.com)
* **MODEL\_EXTENSION functions** — If you deploy Python functions to TabPy as endpoints, you can call them directly (e.g. MODEL\_EXTENSION\_REAL) instead of embedding full logic in the SCRIPT. [help.tableau.com+1](https://help.tableau.com/current/pro/desktop/en-us/r_connection_manage.htm?utm_source=chatgpt.com)
* **Authentication & security** — By default, TabPy is unauthenticated; if used in production or server, secure the connection.

**6. Customer Segmentation Based on Sales Quantiles**

**🎯 Purpose**

Group customers into segments (e.g., Low, Medium, High) based on sales quantiles for targeted marketing or sales strategies.

**🧮 Python Script via SCRIPT\_STR**

SCRIPT\_STR(

"

import pandas as pd

s = pd.Series(\_arg1)

labels = ['Low', 'Medium', 'High']

# Divide into 3 quantiles, label each

segments = pd.qcut(s, q=3, labels=labels)

return segments.astype(str).tolist()

",

SUM([Sales])

)

**📋 Usage**

* Create Customer Segment calculated field with above.
* Put Customer Name on Rows or Detail.
* Put Customer Segment on Color.
* Set **Compute Using**: Customer Name.

**📘 Explanation**

* Divides customers into 3 buckets based on total sales.
* Helps in customer tiering for campaigns or prioritization.

**7. Sales Growth Rate per Region (MoM or YoY)**

**🎯 Purpose**

Calculate month-over-month (MoM) or year-over-year (YoY) sales growth rate per region to monitor performance.

**🧮 Python Script via SCRIPT\_REAL**

SCRIPT\_REAL(

"

import pandas as pd

s = pd.Series(\_arg1)

growth = s.pct\_change().fillna(0) \* 100

return growth.tolist()

",

SUM([Sales])

)

**📋 Usage**

* Put Order Date (continuous month) on Columns.
* Put Region on Color or Rows.
* Use Sales Growth Rate calculated field on Rows.
* Set **Compute Using**: Order Date.
* Use **Restarting Every**: Region.

**📘 Explanation**

* Calculates percent change from previous month/year.
* Identifies accelerating or declining sales trends.

**8. Detect Consistent Sales Increase Over Last N Periods**

**🎯 Purpose**

Flag products or regions with consistent sales increases over last N months — useful for trend spotting.

**🧮 Python Script via SCRIPT\_BOOL**

SCRIPT\_BOOL(

"

import numpy as np

arr = np.array(\_arg1)

# True if every consecutive difference > 0

return [all(np.diff(arr[-5:]) > 0)] \* len(arr)

",

SUM([Sales])

)

**📋 Usage**

* Create Consistent Increase Flag.
* Use in filters or color coding.
* Set **Compute Using**: Order Date.
* Use partitions for product or region.

**📘 Explanation**

* Checks last 5 months if sales are strictly increasing.
* Returns True or False for all rows in partition.

**9. Sales Contribution Rank of Products within Category**

**🎯 Purpose**

Rank products based on sales contribution inside their category for prioritization.

**🧮 Python Script via SCRIPT\_INT**

SCRIPT\_INT(

"

import numpy as np

arr = np.array(\_arg1)

# Rank descending; higher sales = 1

ranks = (-arr).argsort().argsort() + 1

return ranks.tolist()

",

SUM([Sales])

)

**📋 Usage**

* Create Sales Rank in Category.
* Put Category on Rows, Product Name on Columns or Detail.
* Use field to sort or highlight top performers.
* Set **Compute Using**: Product Name.
* Restart partition every Category.

**📘 Explanation**

* Gives rank 1 to highest-selling product, 2 to second highest, etc.
* Supports competitive analysis within category.

**10. Profit Margin Trend Slope (Linear Regression Slope)**

**🎯 Purpose**

Calculate slope of profit margin over time per category or region — indicating improving or deteriorating profitability.

**🧮 Python Script via SCRIPT\_REAL**

SCRIPT\_REAL(

"

import numpy as np

import pandas as pd

y = np.array(\_arg1) # Profit margin values

x = np.arange(len(y)) # Time as integers

# Simple linear regression slope

if len(y) < 2:

return [0] \* len(y)

slope = np.polyfit(x, y, 1)[0]

return [slope] \* len(y)

",

SUM([Profit]) / SUM([Sales])

)

**📋 Usage**

* Create Profit Margin Slope.
* Put Order Date on Columns, Region or Category on Color.
* Use Profit Margin Slope as color or KPI indicator.
* Compute Using: Order Date.
* Restart every: Region or Category.

**📘 Explanation**

* Positive slope = improving margins; negative = margins shrinking.
* Shows trend direction and magnitude in one number.