**Step 1: Understand the Need for TabPy**

Before jumping in, ask:

* Do I need **advanced analytics** beyond Tableau’s native calculations (e.g., machine learning, custom statistics, complex transforms)?
* Do I want to **run Python scripts live** within Tableau dashboards?
* Do I want to **deploy and serve Python models centrally** for multiple users?
* Do I want to **leverage Python libraries** like pandas, numpy, scikit-learn, statsmodels, etc. in Tableau?

If yes, TabPy is the right tool.

**Step 2: Set Up TabPy Server**

TabPy is a Python server that listens for requests from Tableau and executes Python code.

**How to install TabPy**

1. **Install Python** (version 3.x) on your machine/server.
2. Open command prompt / terminal.
3. Run:

pip install tabpy

1. Start TabPy server by running:

tabpy

* By default, TabPy listens on http://localhost:9004.
* You should see a message like:  
  "TabPy is running on port 9004"

**Step 3: Connect Tableau Desktop to TabPy**

1. Open Tableau Desktop.
2. Go to **Help > Settings and Performance > Manage External Service Connection**.
3. Set:
   * **Service:** TabPy/External API
   * **Server:** localhost (or the IP address where TabPy is running)
   * **Port:** 9004 (default)
4. Click **Test Connection** → it should succeed.
5. Click **OK**.

**Step 4: Use SCRIPT Functions in Tableau Calculated Fields**

Tableau provides four SCRIPT\_ functions to embed Python code:

| **Function** | **Output Type** |
| --- | --- |
| SCRIPT\_REAL | Returns real numbers |
| SCRIPT\_INT | Returns integers |
| SCRIPT\_STR | Returns strings |
| SCRIPT\_BOOL | Returns boolean values |

**Step 5: Write Python Code Inside Tableau Calculations**

Example: Calculate moving average of Sales

SCRIPT\_REAL(

"

import numpy as np

sales = \_arg1

arr = np.array(sales)

window = 3

mov\_avg = np.convolve(arr, np.ones(window)/window, mode='valid')

# Pad start to keep same length

pad\_len = len(arr) - len(mov\_avg)

mov\_avg\_padded = np.pad(mov\_avg, (pad\_len, 0), 'constant', constant\_values=(mov\_avg[0],))

return mov\_avg\_padded.tolist()

",

SUM([Sales])

)

* Pass data as arguments (\_arg1, \_arg2, etc.).
* Python code runs in TabPy, returns list of values back to Tableau.
* Tableau plots/uses returned values per data row.

**Step 6: Advanced Use Cases**

* Use **machine learning models** trained externally and deployed in TabPy.
* Use **statistical tests** (e.g., t-test, regression) live.
* Do **complex clustering or segmentation**.
* Generate **predictions, anomaly detection, or trend analysis** live.

**Step 7: Best Practices**

* **Keep data passed to Python sorted and consistent** (e.g., time ascending).
* Use **partitioning (Compute Using)** carefully to control context.
* Minimize data size passed to TabPy for performance.
* Avoid very complex or long-running scripts in dashboards.
* Cache results if needed to improve responsiveness.
* Monitor TabPy server health; consider running in production environment with appropriate resources.

**Step 8: Troubleshooting Tips**

* If connection fails, check firewall/port access.
* Test Python code locally before embedding.
* Use simple prints/logging in TabPy server to debug.
* Ensure Python environment has required packages installed.
* Remember Tableau expects the script to return a list equal in length to input.

**Step 9: Security & Deployment**

* In enterprise, host TabPy on secure server.
* Use SSL/TLS for encryption.
* Control access to TabPy API.
* Use centralized deployment for consistency.

**Summary Flow**

| **Step** | **What Happens** |
| --- | --- |
| **Set up TabPy server** | Python server listens for Tableau requests |
| **Connect Tableau** | Tableau configured to send Python scripts to TabPy |
| **Write SCRIPT\_ calcs** | Python code executed live on data subsets |
| **Receive results** | Python results returned as new Tableau fields |
| **Visualize & act** | Use Python-enhanced fields in dashboards |