

Python in Excel - erste Schritte und Berechnungen

Inhalt

Architektur, Arbeitsweise und technische Grundlagen für Python und Excel	2
Was ist der DataFrame und warum ist er so wichtig?	13
Aggregationen mit groupby() - die Python-Variante der Pivot-Tabelle ...	18
Der Python-Output-Switch - Objekt oder Werte?	25
Empfehlungen aus dem Management-Handbuch	30

Architektur, Arbeitsweise und technische Grundlagen für Python und Excel

Wie Sie Python-Bibliotheken aktivieren und Ihren ersten Python-Code erstellen. Grundlage ist ein Verständnis für die Architektur und Arbeitsweise von Python in Excel.

Zuletzt geändert am 05.03.2026



Ein praxisnaher Einstieg

Stellen Sie sich folgende Situation vor: Sie erhalten aus dem ERP-System eine Datei mit 250.000 Verkaufspositionen. Ziel ist es, für das Management kurzfristig folgende Fragen zu beantworten:

- Wie hoch ist der Umsatz je Region?
- Welche Produktgruppe erzielt den höchsten Deckungsbeitrag?
- Wo liegen die größten Abweichungen zum Vormonat?

In Excel würden Sie:

- eine oder mehrere Pivot-Tabellen erstellen,
- Hilfsspalten für Berechnungen anlegen,
- Formeln kopieren und
- Diagramme erstellen.

Das funktioniert, solange die Datenmenge überschaubar ist und der Prozess nicht regelmäßig wiederholt werden muss. Doch was passiert, wenn

- die Datenmenge wächst,
- die Berechnungslogik komplexer wird,
- der Report monatlich automatisiert erzeugt werden soll oder
- statistische Verfahren ins Spiel kommen?

Genau hier setzt Python in Excel an. Python erweitert Excel um eine strukturierte Analyse-Engine im Hintergrund. Das bedeutet:

- Berechnungen erfolgen nicht mehr zellenweise, sondern spaltenweise und tabellenbasiert.
- Prozesse werden reproduzierbar.
- Logik wird explizit definiert – nicht implizit über Zellbezüge.

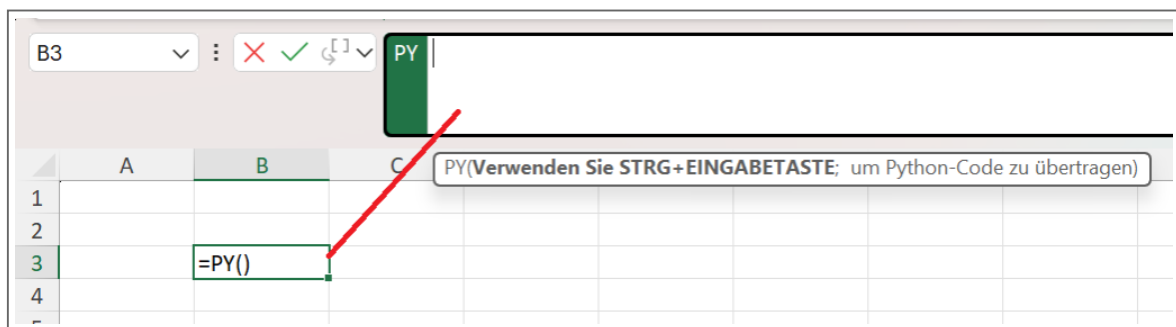
Was passiert technisch bei =PY()?

Wenn Sie in Excel eine normale Formel eingeben, zum Beispiel: **=SUMME(A1:A10)**, berechnet Excel das Ergebnis direkt lokal auf Ihrem Rechner. Die Berechnungslogik ist zellenbasiert, Excel löst globale Abhängigkeiten auf und speichert den Wert in der Zielzelle.

Bei Python in Excel funktioniert das grundlegend anders. Wenn Sie mit Python arbeiten und in einer Excel-Zelle schreiben:

`=PY()`

erstellen Sie keine klassische Excel-Formel, sondern eine Python-Ausführungszelle, einen Codeblock.



Python in Excel als Codeblock

Diese Zelle:

- enthält später einen vollständigen Python-Codeblock,
- führt alle enthaltenen Anweisungen aus und
- gibt das letzte Ergebnis an Excel zurück.

Eine Python-Zelle ist damit eher vergleichbar mit einem kleinen Skript als mit einer einzelnen Zellformel. Innerhalb einer einzigen Zelle können Sie mehrere Schritte definieren:

- Daten einlesen
- Berechnungen durchführen
- Daten aggregieren
- Ergebnis strukturieren

Das ist konzeptionell ein großer Unterschied zur klassischen Excel-Logik.

Wo wird der Python-Code ausgeführt?

Das ist ein zentraler Punkt – besonders für Unternehmen. Der Code läuft nicht lokal auf Ihrem Rechner, sondern in einer isolierten Cloud-Umgebung auf Microsoft Azure. Der technische Ablauf ist:

1. Sie schreiben Python-Code in Excel.
2. Excel sendet diesen Code an eine Azure-Container-Umgebung.
3. Der Code wird dort in einer isolierten Sandbox ausgeführt.
4. Das Ergebnis wird an Excel zurückgesendet.
5. Excel zeigt das Ergebnis als Tabelle oder Objekt an.

Sicherheit und Datenschutz

Gerade im Controlling-Umfeld ist das wichtig. Die Python-Ausführung erfolgt:

- in einer isolierten Container-Umgebung,
- ohne Zugriff auf Ihr lokales Dateisystem,
- ohne Zugriff auf Ihr Unternehmensnetzwerk und
- ohne direkten Internetzugriff.

Das bedeutet: Python in Excel kann keine lokalen Dateien manipulieren und nicht auf interne Systeme zugreifen.

Dennoch gilt: Wenn sensible oder personenbezogene Daten verarbeitet werden, sollte die Nutzung von Python mit IT und Datenschutz abgestimmt werden – insbesondere in regulierten Branchen.

Unterschied zur klassischen Excel-Berechnungslogik

Ein weiterer wichtiger Unterschied betrifft die Denkweise. Excel arbeitet zellenbasiert und mit globaler Abhängigkeitsauflösung. Python in Excel arbeitet blockbasiert und in klar definierten Ausführungseinheiten.

Eine Python-Zelle kann nicht „nachträglich“ auf eine weiter unten liegende Python-Zelle zugreifen, wenn diese noch nicht berechnet wurde.

Python-Zellen werden in Excel nach folgendem Prinzip berechnet: Von links nach rechts und von oben nach unten. Das nennt sich **Row-Major Order**. Das ist wichtig für den strukturellen Aufbau von Modellen.

Warum dieses Architekturverständnis entscheidend ist

Wenn Sie Python in Excel nutzen, wechseln Sie von impliziter Zelllogik zu expliziter Datenlogik. Sie definieren klar:

- welche Daten verwendet werden
- welche Schritte ausgeführt werden
- welches Ergebnis zurückgegeben wird

Das erhöht die Nachvollziehbarkeit, Reproduzierbarkeit und Skalierbarkeit. Ein Python-Code mit =PY() ist daher nicht einfach eine neue Formel – sondern der Einstieg in eine neue Rechenarchitektur innerhalb von Excel.

Bibliotheken in Python in Excel – Was ist bereits vorhanden?

Eine Bibliothek in Python ist eine Sammlung von **fertigem Code**, den andere bereits geschrieben haben, damit man das Rad nicht neu erfinden muss.

Excel-User können diese Sammlungen einfach in ihr Programm importieren, um komplexe Aufgaben wie das Erstellen von Grafiken oder Berechnungen mit nur wenigen Befehlen zu erledigen.

Man kann sie sich wie einen **Werkzeugkasten** vorstellen, aus dem Nutzer für jedes Projekt genau das passende Spezialwerkzeug herausnehmen.

Vorinstallierte Bibliotheken in der Microsoft-Umgebung

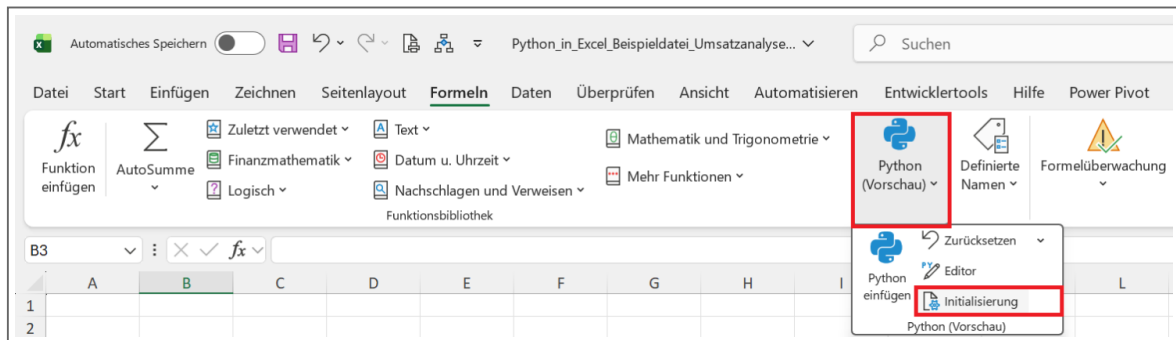
Eine der häufigsten Fragen beim Einstieg lautet: „Muss ich Python oder zusätzliche Pakete erst installieren?“ Die Antwort lautet: Nein.

Python in Excel läuft in einer von Microsoft bereitgestellten Cloud-Umgebung. Dort sind die wichtigsten Analysebibliotheken bereits installiert. Dazu gehören insbesondere:

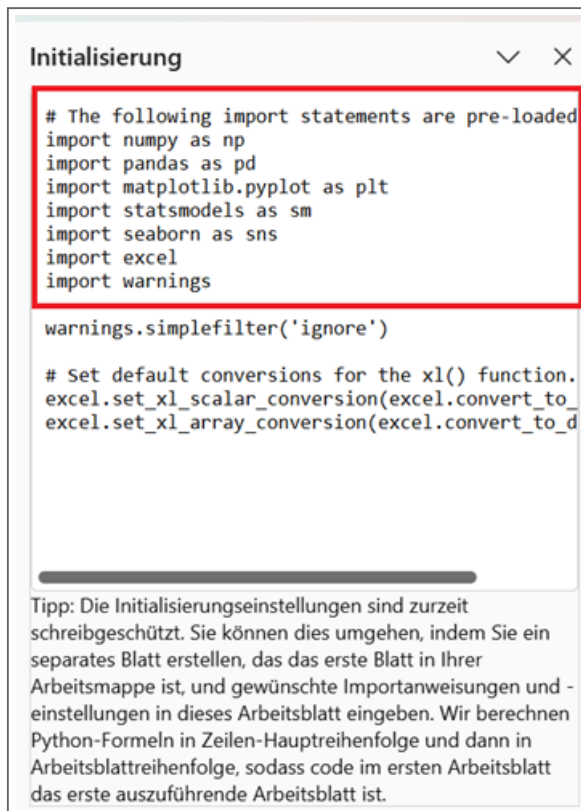
- **Pandas**: Tabellen- und Datenanalyse
- **Numpy**: numerische Berechnungen
- **Matplotlib**: Diagramme
- **Seaborn**: erweiterte Visualisierungen
- **Statsmodels**: statistische Verfahren

Das bedeutet: Sie müssen diese Pakete nicht manuell installieren und keine Abhängigkeiten verwalten.

Die bereits importierten Bibliotheken können Sie sich in Excel anzeigen lassen, indem Sie im Menüband den Befehl Registerkarten **Formel** > Befehl **Python (Vorschau)** > Befehl **Initialisierung** ausführen.

*Python im Excel-Menü*

Sie bekommen daraufhin am rechten Rand den Aufgabenbereich **Initialisierung** angezeigt. Hier können Sie die automatisch importierten Bibliotheken einsehen.

*Bibliotheken in Python*

Warum sollten Sie „import pandas as pd“ schreiben?

Das müssen Sie grundsätzlich nicht machen, da die Pandas-Bibliothek standardmäßig von Excel importiert wird. Es gibt jedoch drei wichtige Gründe, warum es trotzdem gute Praxis ist:

1. Transparenz und Lesbarkeit

Wenn Sie das Excel-Dokument an jemanden schicken oder es nach drei Monaten selbst wieder öffnen, sehen Sie oben sofort in der Zelle: „Ah, hier wird mit Pandas gearbeitet.“ Ohne den expliziten Import müsste man erst im Initialisierungsmenü nachsehen, welche Werkzeuge überhaupt aktiv sind.

2. Kompatibilität (Copy & Paste)

Wenn Sie Ihren Code später außerhalb von Excel nutzen möchten – zum Beispiel in einem anderen Python-Editor wie Jupyter Notebook oder VS Code –, wird der Code dort nicht funktionieren, wenn das „import pandas as pd“ fehlt. Mit der Angabe bleibt der Code portabel.

3. Versionskontrolle und Eindeutigkeit

In der Initialisierung legt Excel fest, welche Version von Pandas genutzt und welches Kürzel (Alias) verwendet wird. Schreiben Sie es selbst, dann haben Sie die volle Kontrolle und stellen sicher, dass genau das passiert, was Sie erwarten.

Unterschied zur lokalen Python-Installation

Wenn Sie Python lokal auf Ihrem Rechner installieren, müssen Sie:

- Python selbst installieren
- Pakete mit pip install nachladen
- Versionen verwalten
- Abhängigkeiten prüfen

Bei Python in Excel entfällt dieser gesamte Aufwand. Das ist insbesondere für Excel-User in Unternehmen mit einer zentralen IT und getrenntem Admin attraktiv. Es sind keine Installationsrechte notwendig und die Umgebung ist für alle Nutzer einheitlich.

Grenzen der vorinstallierten Umgebung

Sie können **keine beliebigen externen Pakete nachinstallieren**. Sie sind auf die Pakete angewiesen, die Microsoft über die Anaconda-Distribution bereitstellt (das sind allerdings über 400 der gängigsten Bibliotheken für Datenanalyse).

Die **Umgebung ist kontrolliert und abgesichert**. Python läuft nicht direkt auf Ihrem Computer, sondern in einer isolierten Schüssel (Container) in der Microsoft Azure Cloud. Das verhindert, dass ein Python-Skript versehentlich (oder absichtlich) auf private Dateien oder das lokale Netzwerk zugreift.

Internetzugriffe sind eingeschränkt. Der Python-Code in Excel kann keine Daten von beliebigen Webseiten nachladen oder E-Mails verschicken. Das ist ein wichtiger Schutzmechanismus für Unternehmen (Governance), damit keine sensiblen Firmendaten unkontrolliert nach außen abfließen

Das ist bewusst so gestaltet – aus Sicherheits- und Governance-Gründen. Für klassische Controlling- und Analyseaufgaben sind die vorhandenen Bibliotheken jedoch mehr als ausreichend.

Excel-Daten mit xl() übergeben – Arbeiten mit der Beispieldatei

Für die folgenden Erläuterungen gibt es eine Excel-Datei mit einem Tabellenblatt:

Umsatzdaten. Dort gibt es die Daten zu:

- Region
- Produktgruppe
- Umsatz
- Kosten

	A	B	C	D
1	Region	Produktgruppe	Umsatz	Kosten
2	Nord	A	120.000,00	80.000,00
3	Nord	B	95.000,00	60.000,00
4	Süd	A	150.000,00	90.000,00
5	Süd	C	110.000,00	70.000,00
6	West	B	100.000,00	65.000,00
7	West	C	85.000,00	50.000,00
8	Ost	A	70.000,00	45.000,00
9	Ost	B	65.000,00	40.000,00
10	Nord	C	105.000,00	75.000,00
11	Süd	B	130.000,00	85.000,00

Beispieldaten in Excel

Daten korrekt an Python übergeben

Angenommen, die Tabelle steht im Bereich A1:D11 und Sie wollen die Daten an eine Excel-Python-Zelle übergeben. Dann schreiben Sie den folgenden Code in die Python-Zelle (F1):

```
import pandas as pd
df = xl("A1:D11", headers=True)
df
```

	A	B	C	D	E	F
1	Region	Produktgruppe	Umsatz	Kosten		[Py] DataFrame
2	Nord	A	120.000,00	80.000,00		
3	Nord	B	95.000,00	60.000,00		
4	Süd	A	150.000,00	90.000,00		
5	Süd	C	110.000,00	70.000,00		
6	West	B	100.000,00	65.000,00		
7	West	C	85.000,00	50.000,00		
8	Ost	A	70.000,00	45.000,00		
9	Ost	B	65.000,00	40.000,00		
10	Nord	C	105.000,00	75.000,00		
11	Süd	B	130.000,00	85.000,00		

Erster Schritt in Python: Daten auswählen und übergeben

Warum headers=True wichtig ist

Standardmäßig versucht Excel zu erkennen, ob die erste Zeile Überschriften enthält. Sicher ist es jedoch, dies explizit zu definieren:

```
df = xl("A1:D11", headers=True)
```

Dadurch wird Zeile 1 als Spaltenname interpretiert und damit ist `df["Umsatz"]` korrekt möglich. Zudem werden keine numerischen Default-Spalten erzeugt. **Explizite Definition** ist in Python grundsätzlich **Best Practice**.

Noch robuster: Arbeiten mit Tabellenobjekten

Besser als feste Zellbereiche ist die Referenzierung einer formatierten Excel-Tabelle. Wenn Sie die Daten als Tabelle formatieren (Strg + T) und beispielsweise „tblUmsatz“ nennen, können Sie schreiben:

```
df = xl("tblUmsatz[#Alle]", headers=True)
```

Vorteile:

- automatische Erweiterung bei neuen Zeilen
- kein Anpassen des Zellbereichs
- stabiler Code bei wachsenden Daten

Was ist mit `df = xl()` passiert?

Mit:

```
df = xl("tblUmsatz[#Alle]", headers=True)
```

haben Sie:

- Excel-Daten in einen DataFrame überführt,
- eine strukturierte, spaltenorientierte Datenstruktur erzeugt und
- die Grundlage für alle weiteren Berechnungen gelegt.

Ab diesem Punkt denken Sie nicht mehr in einzelnen Zellen – sondern in Tabellenobjekten.

Mit **df** geben Sie die eingelesenen Daten wieder in der Python-Zelle F1 aus. Der ausgegebene Code wird im ersten Schritt in einem Python-Objekt (Datenframe) ausgegeben und ist somit auf den ersten Blick nicht sichtbar.

Sie können die Daten direkt im Excel-Blatt visuell zurückgeben lassen, indem Sie die Python-Zelle F1 markieren.

Klicken Sie in der Bearbeitungsleiste auf das Wechsel-Symbol, das sich direkt links neben dem PY befindet. Wählen Sie hier den Eintrag **Excel-Wert** aus.

Python-Ausgabe
Python-Objekt
Excel-Wert

	A	B	C	D	E	F
1	Region	Produktgruppe	Umsatz	Kosten		[Py] DataFrame
2	Nord	A	120.000,00	80.000,00		
3	Nord	B	95.000,00	60.000,00		
4	Süd	A	150.000,00	90.000,00		
5	Süd	C	110.000,00	70.000,00		
6	West	B	100.000,00	65.000,00		
7	West	C	85.000,00	50.000,00		
8	Ost	A	70.000,00	45.000,00		
9	Ost	B	65.000,00	40.000,00		
10	Nord	C	105.000,00	75.000,00		
11	Süd	B	130.000,00	85.000,00		

Funktion: Ausgabe des Python-Objekts als Excel-Wert

Die Daten des DataFrames werden jetzt direkt in das Tabellenblatt geschrieben. Sie können den Inhalt des Datenframes direkt im Excelblatt einsehen.

Python-Ausgabe
Python-Objekt
Excel-Wert

	A	B	C	D	E	F	G	H	I
1	Region	Produktgruppe	Umsatz	Kosten		Region	Produktgr	Umsatz	Kosten
2	Nord	A	120.000,00	80.000,00		Nord	A	120000	80000
3	Nord	B	95.000,00	60.000,00		Nord	B	95000	60000
4	Süd	A	150.000,00	90.000,00		Süd	A	150000	90000
5	Süd	C	110.000,00	70.000,00		Süd	C	110000	70000
6	West	B	100.000,00	65.000,00		West	B	100000	65000
7	West	C	85.000,00	50.000,00		West	C	85000	50000
8	Ost	A	70.000,00	45.000,00		Ost	A	70000	45000
9	Ost	B	65.000,00	40.000,00		Ost	B	65000	40000
10	Nord	C	105.000,00	75.000,00		Nord	C	105000	75000
11	Süd	B	130.000,00	85.000,00		Süd	B	130000	85000

Ergebnis eines Python-Codes

Was ist der DataFrame und warum ist er so wichtig?

Wie Sie mit Python in Excel Ihre Daten bearbeiten und darstellen. Anleitung mithilfe eines einfachen Beispiels und einer ersten Berechnung.

Zuletzt geändert am 05.03.2026



Ein DataFrame ist mehr als eine Tabelle

Wenn Sie mit Python in Excel arbeiten, begegnet Ihnen ein Begriff immer wieder: **DataFrame**. Er ist das Herzstück jeder Datenanalyse in Python.

Auf den ersten Blick sieht ein DataFrame aus wie eine Excel-Tabelle. Technisch ist er jedoch etwas völlig anderes. Ein DataFrame ist eine spaltenorientierte **In-Memory-Datenstruktur**. Das bedeutet:

- Die Daten liegen im Arbeitsspeicher.
- Sie werden spaltenweise organisiert.
- Berechnungen erfolgen optimiert auf ganzen Spalten.

Vergleich: Excel-Zellen vs. DataFrame

Excel speichert jede Zelle einzeln mit Formatierungen, Formeln und Metadaten. Excel ist stark auf Darstellung und Benutzerinteraktion ausgelegt. **Pandas speichert:**

- nur Datenwerte
- spaltenorientiert
- ohne Formatierungslogik
- optimiert für Berechnungen

Das erklärt, warum Python bei größeren Datenmengen oft deutlich schneller ist als klassische Excel-Formeln.

Wichtig: In Excel ist die Zeilennummer fix. In Pandas hat ein DataFrame einen Index. Wenn man Daten sortiert oder filtert, bleibt der Index am Datensatz kleben. Das ist für Excel-Nutzer oft verwirrend, wenn plötzlich die Zeilennummern durcheinander sind.

Einsatz des DataFrame mit einer Beispieldatei

Ausgangspunkt ist der DataFrame, der auf der Basis von Umsatzdaten in einer Excel-Datei angelegt ist und in dieser Anleitung vorgestellt wird: **Architektur, Arbeitsweise und technische Grundlagen für Python und Excel.**

Der Python-Befehl lautet:

```
import pandas as pd
df = xl("tblUmsatz[#Alle]", headers=True)
df
```

Was passiert hier?

- Excel übergibt die Tabelle tblUmsatz an Python.
- Python erzeugt ein DataFrame-Objekt.
- Dieses Objekt wird in Excel angezeigt.

Ab diesem Moment arbeiten Sie nicht mehr mit Zellen, sondern mit einer strukturierten Datenrepräsentation.

	A	B	C	D		F	G	H	I
1	Region	Produktgruppe	Umsatz	Kosten		Region	Produktgr	Umsatz	Kosten
2	Nord	A	120.000,00	80.000,00		Nord	A	120000	80000
3	Nord	B	95.000,00	60.000,00		Nord	B	95000	60000
4	Süd	A	150.000,00	90.000,00		Süd	A	150000	90000
5	Süd	C	110.000,00	70.000,00		Süd	C	110000	70000
6	West	B	100.000,00	65.000,00		West	B	100000	65000
7	West	C	85.000,00	50.000,00		West	C	85000	50000
8	Ost	A	70.000,00	45.000,00		Ost	A	70000	45000
9	Ost	B	65.000,00	40.000,00		Ost	B	65000	40000
10	Nord	C	105.000,00	75.000,00		Nord	C	105000	75000
11	Süd	B	130.000,00	85.000,00		Süd	B	130000	85000

Beispiel: Ausgangsdaten in Excel und DataFrame in Python

Behalten Sie den Überblick mit .head()

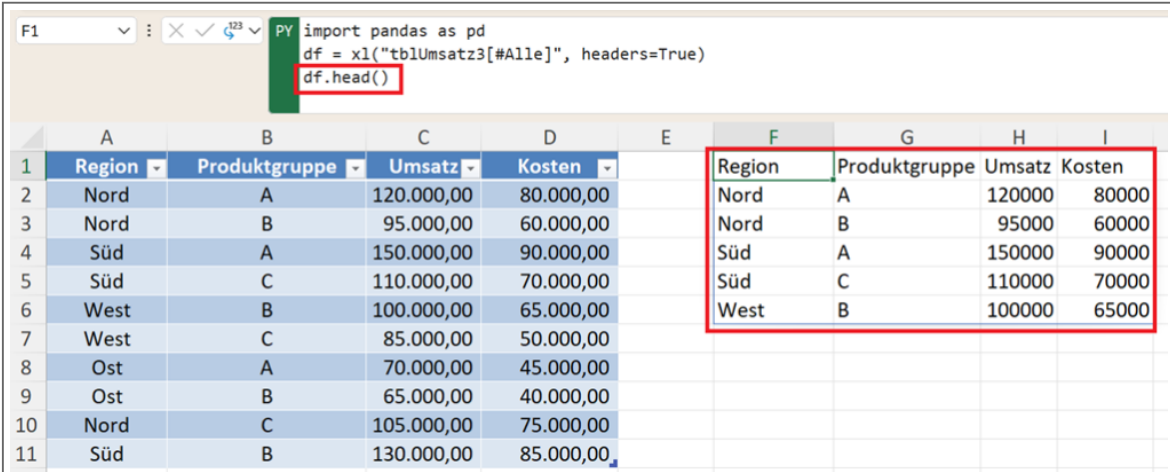
In Excel sind Sie es gewohnt, durch tausende Zeilen zu scrollen, um ein Gefühl für die Daten zu bekommen. In Python ist das oft gar nicht nötig – und bei sehr großen Datensätzen sogar unübersichtlich.

Hier hilft eine der wichtigsten Funktionen in Pandas:

```
df.head()
```

Diese Anweisung zeigt nur die ersten 5 Zeilen Ihres DataFrames an. Warum ist das für Sie nützlich?

- **Schnelle Kontrolle:** Sie sehen sofort, ob die Spaltenüberschriften korrekt erkannt wurden und welche Datentypen vorliegen.
- **Performance:** Anstatt 100.000 Zeilen in Ihr Excel-Sheet zurückzuschreiben, lassen Sie sich nur einen Auszug anzeigen, während Sie Ihre Logik entwickeln.
- **Fokus:** Sie konzentrieren sich auf die Struktur der Daten, nicht auf die Masse.



```

import pandas as pd
df = xl("tblUmsatz3[#Alle]", headers=True)
df.head()

```

	A	B	C	D	E	F	G	H	I
1	Region	Produktgruppe	Umsatz	Kosten		Region	Produktgruppe	Umsatz	Kosten
2	Nord	A	120.000,00	80.000,00		Nord	A	120000	80000
3	Nord	B	95.000,00	60.000,00		Nord	B	95000	60000
4	Süd	A	150.000,00	90.000,00		Süd	A	150000	90000
5	Süd	C	110.000,00	70.000,00		Süd	C	110000	70000
6	West	B	100.000,00	65.000,00		West	B	100000	65000
7	West	C	85.000,00	50.000,00					
8	Ost	A	70.000,00	45.000,00					
9	Ost	B	65.000,00	40.000,00					
10	Nord	C	105.000,00	75.000,00					
11	Süd	B	130.000,00	85.000,00					

Die ersten Datensätze mit `df.head()`

Tipp aus der Praxis: Wenn Sie mehr sehen wollen, schreiben Sie einfach eine Zahl in die Klammer, zum Beispiel für die ersten zehn Zeilen:

```
df.head(10)
```

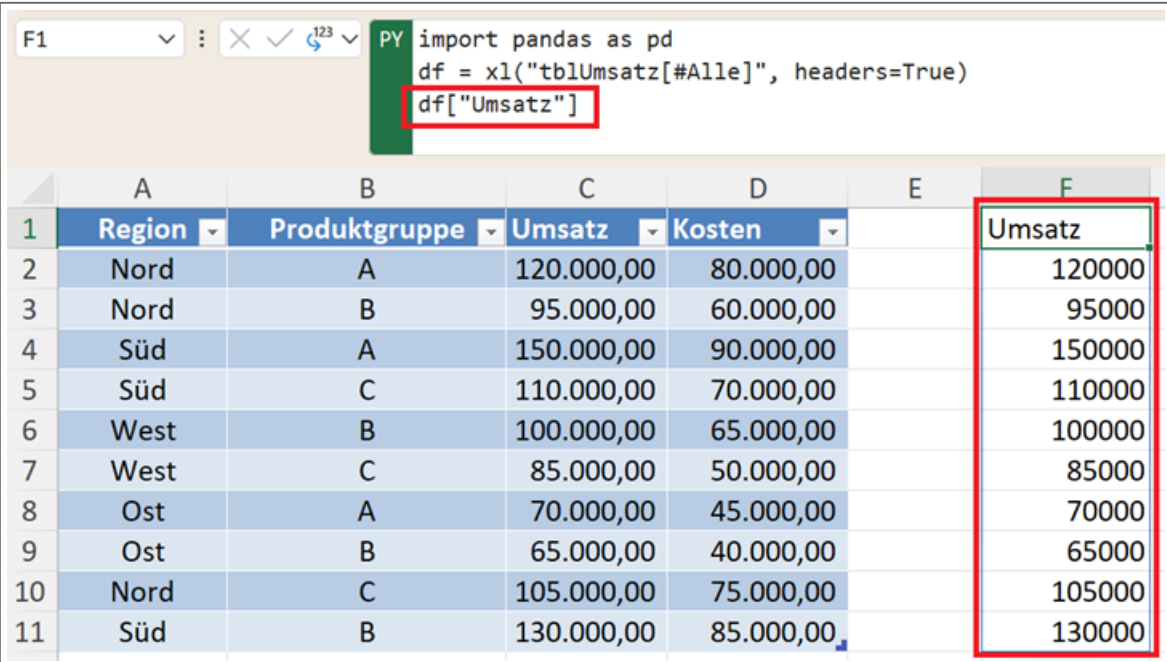
Das Gegenstück dazu ist übrigens **df.tail()**, womit Sie die letzten Zeilen Ihres Datensatzes prüfen können – ideal, um zu sehen, ob am Ende der Tabelle Summenzeilen oder Leerwerte stehen, die Sie eventuell entfernen müssen.

Zugriff auf Spalten des Datensatzes

In Excel würden Sie eine Spalte markieren. Im DataFrame greifen Sie so zu:

```
df["Umsatz"]
```

Das Ergebnis ist keine einzelne Zelle, sondern eine komplette Spalte als Datenobjekt.



	A	B	C	D	E	F
1	Region	Produktgruppe	Umsatz	Kosten		Umsatz
2	Nord	A	120.000,00	80.000,00		120000
3	Nord	B	95.000,00	60.000,00		95000
4	Süd	A	150.000,00	90.000,00		150000
5	Süd	C	110.000,00	70.000,00		110000
6	West	B	100.000,00	65.000,00		100000
7	West	C	85.000,00	50.000,00		85000
8	Ost	A	70.000,00	45.000,00		70000
9	Ost	B	65.000,00	40.000,00		65000
10	Nord	C	105.000,00	75.000,00		105000
11	Süd	B	130.000,00	85.000,00		130000

Auswahl einer Spalte aus dem Datensatz

Stellen Sie sich den DataFrame wie ein Kartenspiel vor: Jede Karte ist eine Spalte. Sie können eine Karte herausnehmen, sie bearbeiten und wieder in den Stapel stecken.

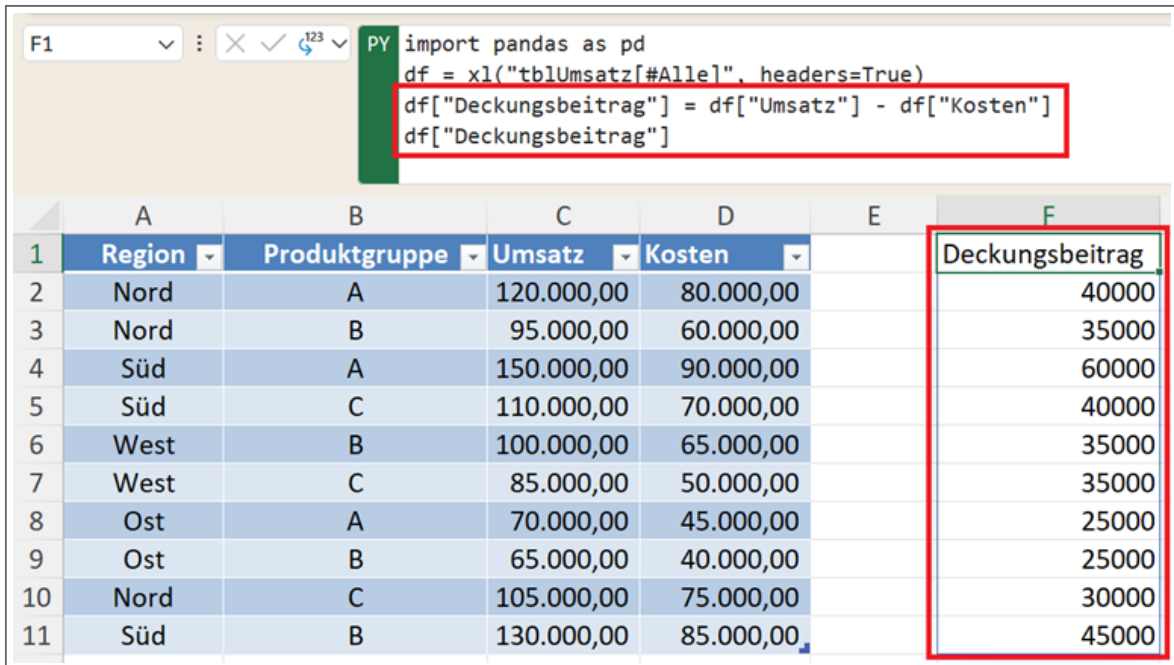
Damit können Sie Operationen direkt auf ganze Spalten anwenden. Beispiel:

```
df["Deckungsbeitrag"] = df["Umsatz"] - df["Kosten"]
```

Hier passiert:

- Die gesamte Umsatz-Spalte wird verarbeitet.
- Die gesamte Kosten-Spalte wird verarbeitet.
- Die Berechnung des Deckungsbeitrags erfolgt vektorisiert.
- Eine neue Spalte wird erzeugt.

Der Vorteil: kein Kopieren von Formeln, kein Ziehen mit der Maus, kein Risiko vergessener Zeilen.



The screenshot shows an Excel spreadsheet with a Python code editor overlay. The code defines a pandas DataFrame 'df' from an Excel table 'tblUmsatz' and calculates the 'Deckungsbeitrag' (Contribution Margin) by subtracting 'Kosten' (Costs) from 'Umsatz' (Revenue). The spreadsheet data is as follows:

	A	B	C	D	E	F
	Region	Produktgruppe	Umsatz	Kosten		Deckungsbeitrag
2	Nord	A	120.000,00	80.000,00		40000
3	Nord	B	95.000,00	60.000,00		35000
4	Süd	A	150.000,00	90.000,00		60000
5	Süd	C	110.000,00	70.000,00		40000
6	West	B	100.000,00	65.000,00		35000
7	West	C	85.000,00	50.000,00		35000
8	Ost	A	70.000,00	45.000,00		25000
9	Ost	B	65.000,00	40.000,00		25000
10	Nord	C	105.000,00	75.000,00		30000
11	Süd	B	130.000,00	85.000,00		45000

Berechnung eines Werts aus den Werten zweier anderer Spalten

Pandas speichert Daten spaltenweise im Speicher. Im Gegensatz dazu speichert Excel jede Zelle als eigenes Objekt. Dieser strukturelle Unterschied ist einer der Hauptgründe für Performancevorteile.

Warum das für Controller relevant ist? Im Controlling arbeiten Sie oft mit:

- großen Transaktionsdaten
- strukturierten Tabellen
- wiederkehrenden Berechnungen

Der DataFrame ist ideal für:

- Aggregationen
- KPI-Berechnungen
- Filterlogik
- Simulationen

Er ist im Grunde eine kleine Datenbank im Arbeitsspeicher.

Aggregationen mit groupby() – die Python-Variante der Pivot-Tabelle

Wie Sie Daten mit Python analysieren und Kennzahlen berechnen mit groupby() und .agg und .sum. Ein Beispiel mit Schritt-für-Schritt-Anleitung.

Zuletzt geändert am 05.03.2026



Praxisfall: Umsatz je Region mit Python ermitteln

Im Controlling lautet eine der häufigsten Fragen: Wie hoch ist der Umsatz oder Deckungsbeitrag je Region?

Ausgangspunkt ist der DataFrame, der auf der Basis von Umsatzdaten in einer Excel-Datei angelegt ist und in dieser Anleitung vorgestellt wird: **Architektur, Arbeitsweise und technische Grundlagen für Python und Excel.**

Der Python-Befehl lautet:

```
import pandas as pd
df = xl("tblUmsatz[#Alle]", headers=True)
df
```

Der Deckungsbeitrag ist berechnet zu:

```
df["Deckungsbeitrag"] = df["Umsatz"] - df["Kosten"]
```

Nun kommt der entscheidende Schritt mit dem Python-Code:

```
df.groupby("Region")["Umsatz"].sum()
```

Was passiert hier genau?

- **groupby("Region")** → Die Daten werden nach Region gruppiert.
- **["Umsatz"]** → Es wird nur die Umsatz-Spalte betrachtet.
- **.sum()** → Der Umsatz wird je Gruppe summiert.

Das Ergebnis ist eine verdichtete Übersicht, die funktional identisch mit einer Pivot-Tabelle ist.

The screenshot shows an Excel spreadsheet with a Python script in the formula bar and a summary table on the right. The script calculates the contribution margin by region. The summary table shows the total sales for each region.

Region	Produktgruppe	Umsatz	Kosten
Nord	A	120.000,00	80.000,00
Nord	B	95.000,00	60.000,00
Süd	A	150.000,00	90.000,00
Süd	C	110.000,00	70.000,00
West	B	100.000,00	65.000,00
West	C	85.000,00	50.000,00
Ost	A	70.000,00	45.000,00
Ost	B	65.000,00	40.000,00
Nord	C	105.000,00	75.000,00
Süd	B	130.000,00	85.000,00

Region	Umsatz
Nord	320000
Ost	135000
Süd	390000
West	185000

Umsatz je Region mit Python in Excel berechnet

Deckungsbeitrag je Region berechnen

Jetzt aggregieren Sie den Deckungsbeitrag als weitere Kennzahl:

```
df.groupby("Region")["Deckungsbeitrag"].sum()
```

Damit erhalten Sie direkt eine Management-Kennzahl ohne Pivot-Konfiguration, manuelle Feldzuweisung und Layout-Anpassung. Die Logik ist vollständig im Code definiert.

Python Code in the formula bar:

```
import pandas as pd
df = xl("tblUmsatz[#Alle]", headers=True)
df["Deckungsbeitrag"] = df["Umsatz"] - df["Kosten"]
df.groupby("Region")["Deckungsbeitrag"].sum()
```

	A	B	C	D	E	F	G
1	Region	Produktgruppe	Umsatz	Kosten		Region	Deckungsbeitrag
2	Nord	A	120.000,00	80.000,00		Nord	105000
3	Nord	B	95.000,00	60.000,00		Ost	50000
4	Süd	A	150.000,00	90.000,00		Süd	145000
5	Süd	C	110.000,00	70.000,00		West	70000
6	West	B	100.000,00	65.000,00			
7	West	C	85.000,00	50.000,00			
8	Ost	A	70.000,00	45.000,00			
9	Ost	B	65.000,00	40.000,00			
10	Nord	C	105.000,00	75.000,00			
11	Süd	B	130.000,00	85.000,00			

Deckungsbeitrag je Region mit Python in Excel berechnet

Ein wichtiger technischer Punkt: Das Ergebnis ist eine Series

Wenn Sie in Excel eine Pivot-Tabelle erstellen, erhalten Sie immer eine Tabelle mit Zeilen und Spalten. In Python passiert nach einer einfachen Aggregation wie `.sum()` etwas Besonderes: Das Ergebnis ist kein gewöhnlicher Datensatz (DataFrame), sondern ein **Series-Objekt**.

Das bedeutet:

- „Region“ wird zum Index
- Die Spaltenbezeichnung ist nicht explizit sichtbar

Was bedeutet das für Sie?

Stellen Sie sich eine Series als eine einzelne Datenspalte vor, die jedoch eine Besonderheit hat: Die Beschriftungen (im Beispiel die „Region“) sind nicht mehr Teil der Daten, sondern sie sind zum Index geworden.

Warum ist das ein Problem für Excel?

Wenn Sie dieses Series-Objekt direkt an eine Excel-Zelle zurückgeben, fehlen oft die Spaltenüberschriften oder die Zuordnung der Zeilenbeschriftungen wirkt „verschoben“.

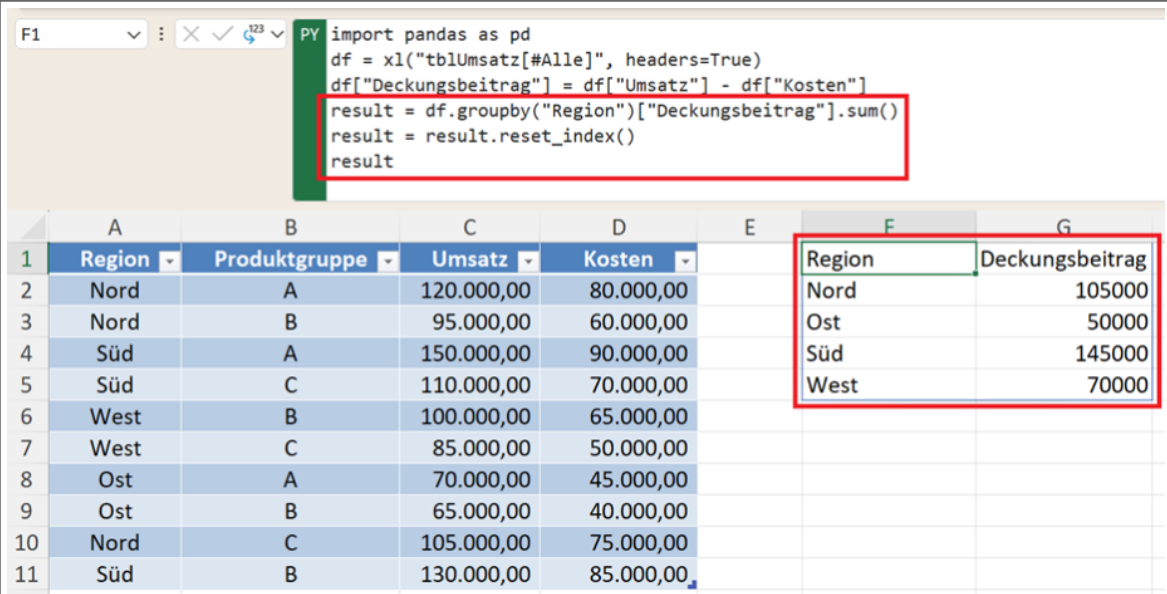
Damit Excel die Daten wieder als saubere, zweidimensionale Tabelle erkennt, nutzen Sie einen einfachen Trick. Sie wandeln den Index mit **reset** wieder in eine normale Datenspalte um.

```
result = df.groupby("Region")["Deckungsbeitrag"].sum()
result = result.reset_index()
result
```

Warum ist reset_index() wichtig?

- **Die Rückverwandlung:** Der Index „Region“ wird wieder zu einer ganz normalen Datenspalte.
- **Klare Struktur:** Das Ergebnis ist nun wieder ein DataFrame.
- **Saubere Übergabe:** Excel kann die Daten nun perfekt mit korrekten Überschriften in die Zellen schreiben – genau so, wie Sie es von einer Pivot-Tabelle erwarten.

Wichtig: Immer, wenn Sie nach einem **groupby()** nur eine einzige Spalte berechnen, erzeugt Pandas eine Series. Mit `.reset_index()` machen Sie daraus wieder eine „excel-taugliche“ Tabelle.



The screenshot shows a Jupyter Notebook interface. The top part contains a Python code cell with the following code:

```
import pandas as pd
df = xl("tblUmsatz[#Alle]", headers=True)
df["Deckungsbeitrag"] = df["Umsatz"] - df["Kosten"]
result = df.groupby("Region")["Deckungsbeitrag"].sum()
result = result.reset_index()
result
```

The bottom part of the screenshot shows the output of the code as an Excel table. The table has two columns: 'Region' and 'Deckungsbeitrag'. The data is as follows:

Region	Deckungsbeitrag
Nord	105000
Ost	50000
Süd	145000
West	70000

Mit `reset_index()` aus einer Python-Series eine korrekte Tabelle erzeugen

Mehrdimensionale Aggregation (Ausblick)

Sie können auch mehrere Dimensionen kombinieren und die Addition der jeweiligen Umsätze direkt mit `reset_index()` verbinden:

```
df.groupby(["Region", "Produktgruppe"])["Umsatz"].sum().reset_index()
```

Das entspricht einer Pivot-Tabelle mit:

- Region in Zeilen
- Produktgruppe als Untergliederung
- Umsatz als Summe

Und das in einer einzigen Codezeile.

The screenshot shows an Excel interface with a Python code editor in the formula bar. The code is:

```
import pandas as pd
df = xl("tblUmsatz[#Alle]", headers=True)
df["Deckungsbeitrag"] = df["Umsatz"] - df["Kosten"]
df.groupby(["Region", "Produktgruppe"])["Umsatz"].sum().reset_index()
```

Below the code, a pivot table is displayed with the following data:

Region	Produktgruppe	Umsatz
Nord	A	120000
Nord	B	95000
Nord	C	105000
Ost	A	70000
Ost	B	65000
Süd	A	150000
Süd	B	130000
Süd	C	110000
West	B	100000
West	C	85000

Gruppierung nach zwei Kriterien mit Python in Excel

Warum groupby() so mächtig ist

`groupby()` ist eines der wichtigsten Werkzeuge in Pandas. Sie können damit:

- Summen bilden
- Durchschnittswerte berechnen
- Minimum oder Maximum bestimmen
- Mehrere Kennzahlen gleichzeitig aggregieren

Beispiel:

```
df.groupby("Region").agg({
    "Umsatz": "sum",
    "Kosten": "sum",
    "Deckungsbeitrag": "sum"
}).reset_index()
```

Damit erzeugen Sie eine komplette Management-Übersicht.

	A	B	C	D		F	G	H	I
1	Region	Produktgruppe	Umsatz	Kosten		Region	Umsatz	Kosten	Deckungsbeitrag
2	Nord	A	120.000,00	80.000,00		Nord	320000	215000	105000
3	Nord	B	95.000,00	60.000,00		Ost	135000	85000	50000
4	Süd	A	150.000,00	90.000,00		Süd	390000	245000	145000
5	Süd	C	110.000,00	70.000,00		West	185000	115000	70000
6	West	B	100.000,00	65.000,00					
7	West	C	85.000,00	50.000,00					
8	Ost	A	70.000,00	45.000,00					
9	Ost	B	65.000,00	40.000,00					
10	Nord	C	105.000,00	75.000,00					
11	Süd	B	130.000,00	85.000,00					

DataFrame in Python mit mehreren aggregierten Kennzahlen

Fazit

In Excel konfigurieren Sie eine Pivot-Tabelle visuell. In Python definieren Sie die Aggregationslogik explizit. Das hat Vorteile:

- Reproduzierbarkeit
- Dokumentation
- Wiederverwendbarkeit
- Erweiterbarkeit

Gerade bei wiederkehrenden Reports ist das ein großer Mehrwert.

Mit groupby() haben Sie:

- eine strukturierte Alternative zur Pivot-Tabelle,
- eine skalierbare Aggregationslogik und
- die Basis für nahezu alle Controlling-Analysen.

Und vor allem: Sie haben gelernt, wie Python Daten nicht visuell, sondern strukturell verarbeitet.

Der Python-Output-Switch – Objekt oder Werte?

Worum geht es beim ‚Output-Switch‘ in Python? Unterschiede und Zusammenspiel zwischen Python-Objekt und Excel-Wert und wann welcher Modus Ihr Projekt technisch und organisatorisch am besten voranbringt.

Zuletzt geändert am 17.03.2026

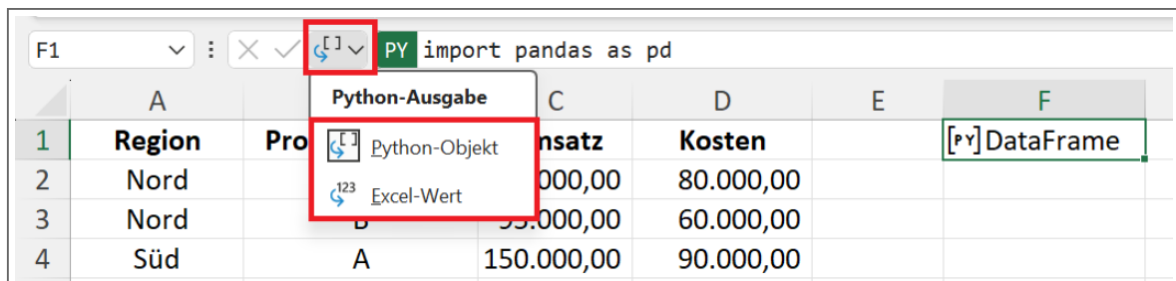


Zwei Darstellungsmodi für das Python-Ergebnis

Haben Sie einen Python-Code in einer Excel-Zelle erstellt, sehen Sie das Python-Ergebnis als kompaktes **Python-Objekt** oder als **tabellarisches Ergebnis (Excel-Wert)** direkt im Tabellenblatt.

Doch hinter dem kleinen Symbol, das Ihnen nach dem Ausführen einer Python-Zelle begegnet, steckt mehr als nur eine rein optische Entscheidung. Es ist das zentrale Steuerungselement, mit dem Sie bestimmen, wie Ihre Analyse mit dem Rest Ihres Modells kommuniziert.

In Excel haben Sie über das Dropdown-Menü in der Bearbeitungsleiste (neben der Formel) die Wahl zwischen zwei Ausgabeformaten:

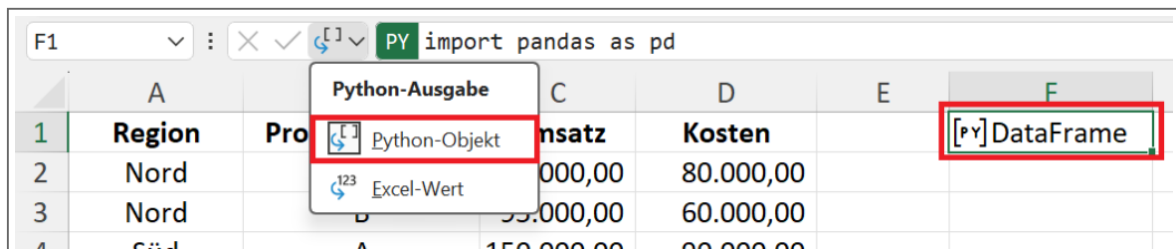


Zwei Ausgabeformate für Python in Excel

Ausgabe als Python-Objekt (Standard)

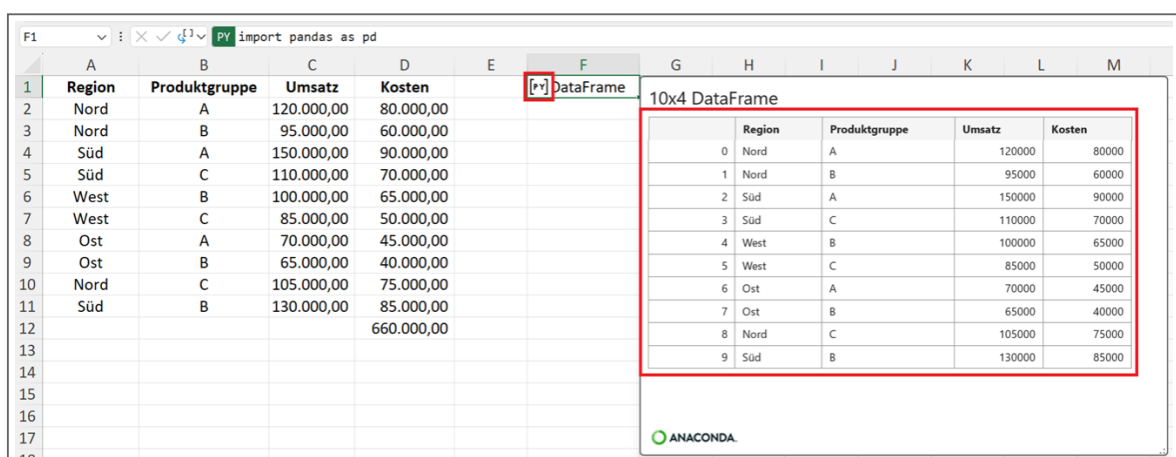
Bei der Ausgabe als Python-Objekt zeigt die Zelle nur [PY]DataFrame. Klickt man auf das Icon, öffnet sich eine Vorschau-Karte, die die Daten anzeigt, ohne das Tabellenblatt zu überlagern. Das ist ideal für Zwischenschritte in der Datenbereinigung. Denn:

- Das Ergebnis wird als Rich-Value-Karte in einer einzelnen Zelle gespeichert.
- Es verhält sich wie ein Container: Ein ganzer DataFrame „lebt“ in einer Zelle.
- Vorteil: Die Daten bleiben im Speicher von Python performant und können von anderen Python-Zellen direkt als Objekt, zum Beispiel über den DataFrame (df), weiterverarbeitet werden.



Ausgabe als Python-Objekt und DataFrame für die Ergebnisse

Hinweis: Die Vorschau-Karte können Sie sich auch mit der Tastenkombination **Strg + Umschalt + F5** anzeigen lassen. Klicken Sie hierzu auf die entsprechende Python-Zelle und führen Sie dann die Tastenkombination aus.



Vorschau auf das Ergebnis des Python-Codes

Ausgabe der Excel-Werte

Das Objekt wird „ausgeschüttet“ (Spilling). Ein DataFrame füllt dann automatisch einen Bereich aus Zeilen und Spalten aus.

Vorteil: Die Daten sind für Standard-Excel-Funktionen (SVERWEIS, Bedingte Formatierung, Filter etc.) direkt greifbar.

	A	B	C	D	E	F	G	H	I
1	Region	Produktgruppe	Umsatz	Kosten		Region	Produktgruppe	Umsatz	Kosten
2	Nord	A	120.000,00	80.000,00		Nord	A	120000	80000
3	Nord	B	95.000,00	60.000,00		Nord	B	95000	60000
4	Süd	A	150.000,00	90.000,00		Süd	A	150000	90000
5	Süd	C	110.000,00	70.000,00		Süd	C	110000	70000
6	West	B	100.000,00	65.000,00		West	B	100000	65000
7	West	C	85.000,00	50.000,00		West	C	85000	50000
8	Ost	A	70.000,00	45.000,00		Ost	A	70000	45000
9	Ost	B	65.000,00	40.000,00		Ost	B	65000	40000
10	Nord	C	105.000,00	75.000,00		Nord	C	105000	75000
11	Süd	B	130.000,00	85.000,00		Süd	B	130000	85000
12				660.000,00					

Ausgabe als Ergebnistabelle mit den Excel-Werten

Die Tabelle erscheint im Tabellenblatt. Ändern sich die Quelldaten, berechnet Python das Ergebnis neu und der Bereich passt sich dynamisch an.

Tipp: Nutzen Sie den Shortcut **Strg + Alt + Umschalt + M**, um schnell zwischen Objekt- und Wert-Ausgabe umzuschalten. Markieren Sie hierzu die betreffende Python-Zelle und drücken Sie anschließend die Tastenkombination.

Wann sollte man welche Variante nutzen?

Welche Variante, Python-Objekt oder Excel-Wert, am besten geeignet ist, leitet sich vom **Anwendungszweck** ab:

- **Zwischenberechnungen** → Python-Objekt → Hält das Tabellenblatt sauber und spart Performance.
- **Finale Berichte** → Excel-Werte → Kollegen können die Zahlen ohne Python-Kenntnisse lesen.

- **Große Datenmengen** → Python-Objekt → Excel muss nicht tausende Zellen rendern, was die Datei schneller macht.
- **Dashboards** → Excel-Werte → Ermöglicht die Nutzung von Excel-Charts basierend auf den Python-Daten.

Technischer Hintergrund und Performance

Solange Sie im Modus Python-Objekt bleiben, findet kein Datentransfer in das klassische Excel-Tabellenblatt statt. Dies ist bei Datensätzen mit 100.000 oder mehr Zeilen entscheidend:

- Ein Objekt belegt nur eine Zelle.
- Erst beim Umschalten auf Excel-Werte muss Excel den gesamten Bereich im Arbeitsspeicher des Rechners verwalten.

Typischer Fehler: #ÜBERLAUF! (#SPILL!)

Wenn Sie auf den Modus Excel-Werte umstellen, versucht Python, die Daten in den umliegenden Bereich zu schreiben. Das Problem: Steht ein einziger Wert (oder auch nur ein Leerzeichen) im Weg, erscheint **#ÜBERLAUF!**

	A	B	C	D	E	F	G	H	I
1	Region	Produktgruppe	Umsatz	Kosten		#ÜBERLAUF!			
2	Nord	A	120.000,00	80.000,00					
3	Nord	B	95.000,00	60.000,00					
4	Süd	A	150.000,00	90.000,00					
5	Süd	C	110.000,00	70.000,00					
6	West	B	100.000,00	65.000,00					
7	West	C	85.000,00	50.000,00					
8	Ost	A	70.000,00	45.000,00					
9	Ost	B	65.000,00	40.000,00					
10	Nord	C	105.000,00	75.000,00					
11	Süd	B	130.000,00	85.000,00					
12				660.000,00					

Die Ausgabe der Excel-Werte ist wegen fehlendem Platz nicht möglich: **#ÜBERLAUF!**

Lösung: Markieren Sie die Python-Zelle. Excel zeigt Ihnen mit einer dünnen blauen Linie den benötigten Bereich an. Löschen Sie die blockierenden Daten in diesem Bereich.

Fazit: Die strategische Brücke

Der Output-Switch ist das Steuerungsinstrument für Ihre Workflow-Hygiene. Er erlaubt es Ihnen, Python als leistungsstarke Engine im Hintergrund zu nutzen (Objekt-Modus) und nur die relevanten Ergebnisse gezielt auf dem Tabellenblatt auszuspielen (Werte-Modus).

Empfehlungen aus dem Management-Handbuch

Python in Excel – Grundlagen

Was Python in Excel leisten kann im Vergleich zu den bordeigenen Mitteln von Excel und VBA. Mit einer Schritt-für-Schritt-Anleitung zur Aktivierung der Python-Funktion in Excel und mit ersten Beispielen, wie Python-Code aufgebaut ist.

<https://www.business-wissen.de/id/kapitel/299/>

Python in Excel für die Datenanalyse

Wie Sie Datenanalyse mit Python und Pandas systematisch durchführen, Datenstrukturen verstehen, aussagekräftige Kennzahlen berechnen und schnell anschauliche Diagramme erstellen. Mit praxisnahen Beispielen und Schritt-für-Schritt-Anleitungen zur Bereinigung, Analyse und Aufbereitung von Datensätzen.

<https://www.business-wissen.de/id/kapitel/318/>

Nutzen Sie als
Premium-Mitglied
alle
Handbuch-Kapitel
mit mehr als
3.000 Checklisten und Excel-Vorlagen

Jetzt anmelden

www.business-wissen.de/anmelden/

Impressum

b-WISE GmbH Business Wissen Information Service
Bismarckstraße 21
76133 Karlsruhe
DEUTSCHLAND

service@business-wissen.de

Telefon +49 721 18397-0

Copyright 2026, b-wise GmbH, All Rights Reserved