OPEN CONTRACTING PARTNERSHIP

DATA USE GUIDE

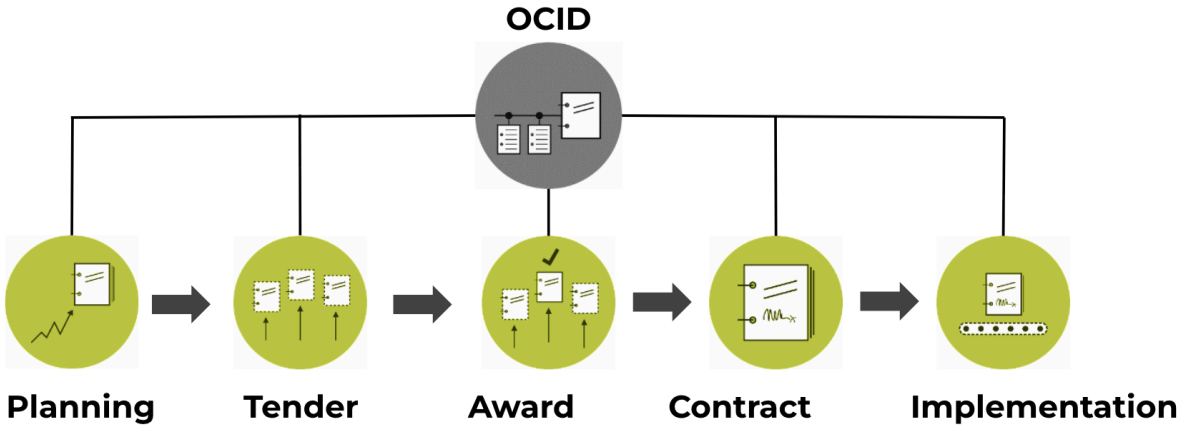
Using OCDS data
A practical guide

This guide is written for data analysts who want to use standardized open data to analyze public procurement processes, for example, to calculate procurement indicators related to market opportunity, internal efficiency, value for money, public integrity and service delivery. It requires some technical knowledge of Python.

1. Understanding OCDS flattened data

The Open Contracting Data Standard (OCDS) is a free, non-proprietary open data standard for public contracting. It describes how to publish data and documents about contracting processes, covering all stages from planning to implementation. You can read the full documentation, or learn more using our learning videos.

OCDS organizes procurement data around a **contracting process**, and each contracting process has a unique identifier called an **ocid**.



Within each process, data is organized into stages: **planning**, **tender**, **awards**, and **contracts**. Each stage can contain arrays of objects: for example, a single tender can have multiple lots and multiple awards, and each award can have its own supplier.
That's why OCDS is structured as JSON, allowing nested arrays and objects. To make it accessible in spreadsheet tools, we use the tool *flatterer* to flatten the data into multiple tables, where each array in the JSON becomes a separate table.

These tables are then published in OCP's Data Registry as Excel and CSV files: each table becomes a sheet in the Excel workbook or an individual CSV file. JSON and CSV files are available for all publishers. Excel files are also available for most; the exceptions are datasets too large for Excel to handle.

The following diagram illustrates how OCDS JSON data is flattened:

In the flattened OCDS version, each contracting process is represented as a single row in the main table, containing most of the information from the tender stage. Each table is linked to its parent table via the [table_name]_id column.

The number of tables you'll find for each publisher depends on which fields they publish. For example, for a publisher that has a tender.documents array in the tabular version of OCDS, there will be a tender_documents table containing this information. In the following table, we list those most commonly used for analysis:

Table	Description	Links to main via	Links to parent via	Some of the key columns
main	One row per contracting process. It includes information from the tender stage	—	—	ocid, date, tender_* fields
awards	Awards issued for the tenders. Each row is an award.	main_ocid	—	id, status, value_amount
awards_suppliers	Suppliers per award. Each row is a supplier.	main_ocid	awards_id → awards_id	name, identifier
parties	Organizations referenced. Each row is a party linked to a specific procurement process.	main_ocid	To link to suppliers or buyer information, besides the main_ocid the roles should be used	name, roles, identifier, details_scale
contracts	Signed contracts. Each row is a contract	main_ocid	awardID → awards.id	id, dateSigned, value_amount
*_items	Items information including the item name, description, ids. Each row is an item	main_ocid	*_id → *_id	
*_documents	Documents from different stages of the process. Each row is a document	main_ocid	—	documentType, datePublished

2. How to link OCDS flattened data

To run an analysis that calculates common procurement indicators, you need to combine information from different tables. All joins use main_ocid as the primary link to the main table, plus [parent_table]_id to link nested tables to their direct parent.

The linking rules are:

1. **Every child table** links back to the main table via the main_ocid column, which matches main.ocid.
2. **Nested child tables** link to their parent table via a [parent_table_name]_id column. For example, awards_suppliers.awards_id matches awards.id, and contracts_documents.contracts_id matches contracts.id.

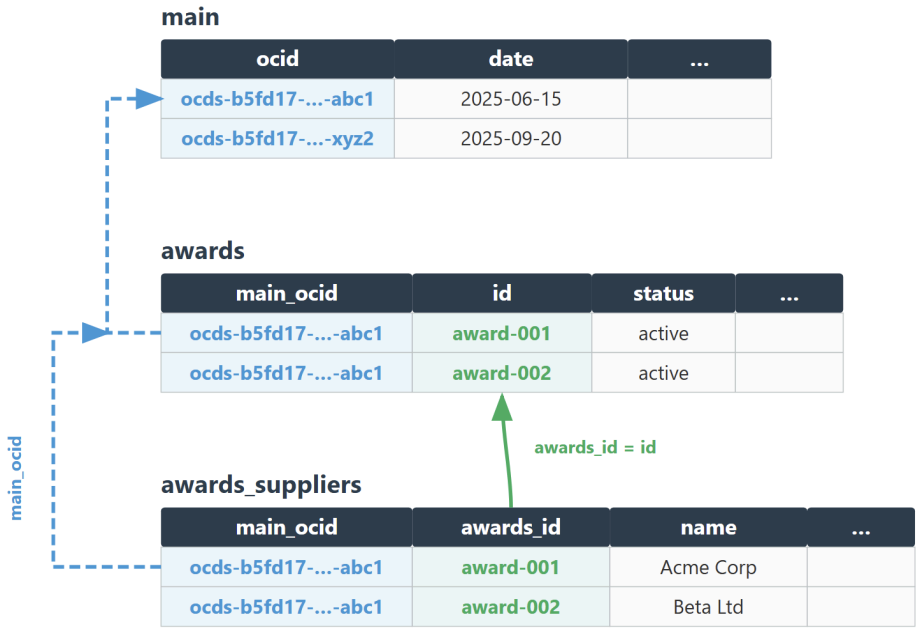


Image generated by Claude

Below, we provide examples of how to do common joins that are useful for analysis, using Python:

1. Linking contracts to procurement methods

Contracts don't directly include the procurement method, as this information lives in the main table under tender_procurementMethod (a field that uses the standardized OCDS codelist) and tender_procurementMethodDetails (local procurement methods).

To analyze contracts by method, join on main_ocid:

```python
#Merge the contracts dataframe with the procurement method from the main sheet.
contracts_full = contracts.merge(
  main[['ocid', 'tender_procurementMethod', 'tender_procurementMethodDetails']],
  left_on='main_ocid', right_on='ocid',
  how='inner', suffixes=('', '_main')
)
```

2. Linking contracts to suppliers and getting the supplier details

One useful join is connecting a contract to its suppliers' details, such as whether they are SMEs (*Note: not all publishers publish the parties details scale field)*. This requires three tables: awards_suppliers, contracts and parties. If you require more variables, like the tender_procurementMethodDetails, you might need to join with the main sheet.

Here is the join pattern:

1. **Step 1: Link suppliers to their party scale** to identify whether it's an SME. First, in the parties sheet, filter only rows where the roles contain 'supplier'. Then you can link the awards_suppliers table with the details_scale using the main_ocid variables and the name or id. In this example, we will join by id.

2. **Step 2: Join contracts to the award's suppliers**, using the main_ocid variable in the contracts and awards_suppliers sheets and contracts.awardID and awards_suppliers.awards_id. In this dataset, a contract can have multiple suppliers, so this join duplicates contract rows that have multiple suppliers and adds one row for each supplier.

```python
# Step 1: Link suppliers to their party scale, to identify if it's an SME
# First, in the parties sheet, filter only rows where the roles contain 'supplier'
suppliers_parties = parties[parties['roles'].str.contains('supplier', na=False)]
# Parties are matched by name within the same contracting process (main_ocid)
award_sup_with_scale = awards_suppliers.merge(
  suppliers_parties[['main_ocid', 'name', 'details_scale']],
  on=['main_ocid', 'name'],
  how='left'
)

# Step 2: Join contracts to the award's suppliers
# Using main_ocid and contracts.awardID = awards_suppliers.awards_id
#This assumes a contract has a single supplier, if a contract has multiple suppliers, then this join duplicates contract rows that have multiple suppliers (one row per supplier)
contracts_with_scale = contracts.merge(
  award_sup_with_scale[['main_ocid', 'awards_id', 'name', 'details_scale']],
  left_on=['main_ocid', 'awardID'],
  right_on=['main_ocid', 'awards_id'],
  how='left'
)
```

3. Which date to use?

One of the benefits of the OCDS format is that it can be updated whenever new information is added throughout the procurement process using releases, making it easier to track the full procurement cycle.

The date field in the main sheet indicates the latest date a procedure was updated (date of the latest release).

You can use this field as your reference date or, depending on the metric you want to calculate, you might want to use other date fields for your analysis. For instance, if you are analyzing contracts over time, it's best to use the contracts.dateSigned field. If you want to count the number of tenders over time, you could use the date the tender opens for submissions, tender.tenderPeriod.startDate.

4. Contact

If you want to see an example of using OCDS data from the Data Registry, you can read our blog on using the UK FTS dataset.

Reach out to us if you have any questions. **We're ready to help!**