# Grouping to a numerator and a denominator

groupby, named aggregation, and the two defaults that silently change a denominator — dropped missing groups and unobserved categories. Producing both halves of an indicator in one operation so they cannot drift.

---

Pierre Robentz Cassion

Lesson 6 of 8

Python for Programme Data — Working the table

# What this lesson covers

- An indicator is two numbers
- Named aggregation
- Two defaults that change a denominator
- Grouping by more than one key
- transform: a group statistic on every row
- apply, and why to avoid it
- Pivot tables read better for a report
- Writing the result out
- What comes next

# An indicator is two numbers — In Python (cont.)

```
import pandas as pd

muac = read_register(RAW / "muac-screening-artibonite-2024.v1.csv")

measured = muac["muac_mm"].notna()
gam = measured & (muac["muac_mm"] < 125)

by_commune = (
    muac.assign(measured=measured, gam=gam)
    .groupby("commune")
    .agg(
        screened=("child_id", "size"),
        measured=("measured", "sum"),
        gam_cases=("gam", "sum"),
    )
)
```

# An indicator is two numbers — In Python (cont.)

```
by_commune["gam_rate"] = (by_commune["gam_cases"] / by_commune["measured"]).round(3)
```

# An indicator is two numbers — Example

| commune | screened | measured | gam_cases | gam_rate |
|---|---|---|---|---|
| Gros-Morne | 361 | 351 | 48 | 0.137 |
| Anse-Rouge | 229 | 223 | 30 | 0.135 |
| L-Estere | 189 | 187 | 18 | 0.096 |
| Terre-Neuve | 241 | 235 | 21 | 0.089 |

# Named aggregation — In Python

```
.agg(
    output_name=("input_column", "function"),
    ...
)
```

# Named aggregation — In Python

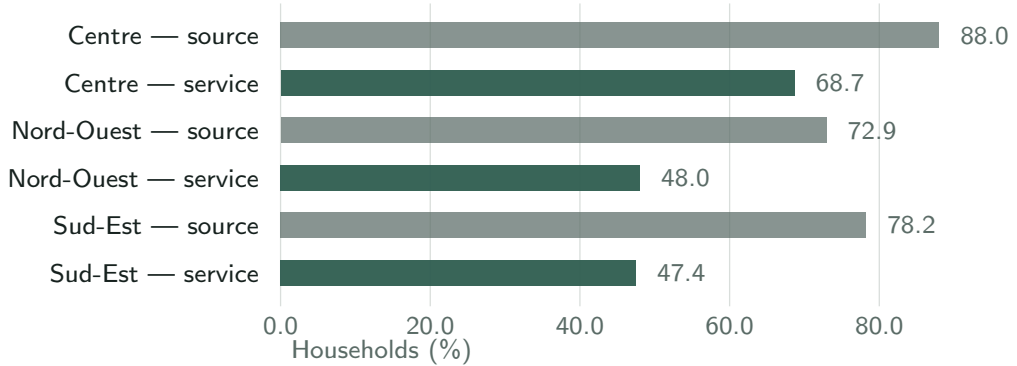
```
muac.groupby("commune")["muac_mm"].mean()                # one column, one function
muac.groupby("commune").agg({"muac_mm": ["mean", "std"]})  # MultiIndex columns
```

# Named aggregation — In Python

```
def share_below(series, threshold=125):
    valid = series.dropna()
    return (valid < threshold).mean() if len(valid) else float("nan")

muac.groupby("commune").agg(
    gam_rate=("muac_mm", share_below),
    n=("muac_mm", "count"),
)
```

# Two defaults that change a denominator




Improved water source against basic service, by district. Counting sources overstates coverage everywhere; the gap is households whose source is improved and more than thirty minutes away.

# Two defaults that change a denominator

- Missing group keys are dropped

# Two defaults that change a denominator — In Python

```
d = pd.DataFrame({"g": ["a", None, "a"], "v": [1, 2, 3]})

d.groupby("g")["v"].sum()                  # {'a': 4}
d.groupby("g", dropna=False)["v"].sum()    # {'a': 4, nan: 2}
```

# Two defaults that change a denominator — In Python

```
assert muac["commune"].notna().all(), "rows with no commune would be dropped"
```

# Two defaults that change a denominator

- Unobserved categories reappear as empty rows

# Two defaults that change a denominator — In Python

```
c = pd.DataFrame({"k": pd.Categorical(["x", "y"], categories=["x", "y", "z"]),
                  "v": [1, 2]})

len(c.groupby("k", observed=True)["v"].sum())    # 2
len(c.groupby("k", observed=False)["v"].sum())   # 3 -- includes an empty "z"
```

## Two defaults that change a denominator

- Pass it explicitly — The default changed between pandas versions, and a script that relies on it produces a different...

# Grouping by more than one key — In Python

```
by_commune_month = (
    muac.assign(month=muac["screening_date"].dt.to_period("M"), gam=gam)
    .groupby(["commune", "month"], observed=True)
    .agg(measured=("muac_mm", "count"), gam_cases=("gam", "sum"))
)
```

# Grouping by more than one key — In Python

```
by_commune_month.loc["Gonaives"]          # one commune, all months
by_commune_month.reset_index()            # back to flat columns
```

# Grouping by more than one key — In Python

```
wide = by_commune_month["gam_cases"].unstack("month", fill_value=0)
```

# transform: a group statistic on every row — In Python

```
muac["commune_mean"] = muac.groupby("commune")["muac_mm"].transform("mean")
muac["vs_commune"] = muac["muac_mm"] - muac["commune_mean"]
```

# apply, and why to avoid it — In Python

```python
# Works, but slow, and returns something whose shape depends on the function.
muac.groupby("commune").apply(lambda g: g["muac_mm"].mean(), include_groups=False)
```

# Pivot tables read better for a report — In Python

```
pd.crosstab(muac["commune"], muac["outcome"])

pd.crosstab(
    muac["commune"], muac["outcome"],
    values=muac["muac_mm"], aggfunc="mean",
).round(1)
```

# Writing the result out — In Python

```
OUTPUTS = PROJECT / "outputs" / "tables"
OUTPUTS.mkdir(parents=True, exist_ok=True)

by_commune.reset_index().to_csv(OUTPUTS / "gam_by_commune.csv", index=False)
```

# Writing the result out — In Python

```
definitions = pd.DataFrame([
    ("GAM", "MUAC < 125 mm or bilateral pitting oedema",
     "children with a MUAC measurement or a recorded oedema assessment"),
], columns=["indicator", "numerator", "denominator"])

definitions.to_csv(OUTPUTS / "definitions.csv", index=False)
```

## What comes next

- The table aggregates correctly and both halves of every rate come from one operation.

## Read the full lesson, with runnable code

https://data-analysis.cassion.dev/courses/python-for-programme-data/python-06-grouping-and-aggregation