

# Data Operations Reference Sheet

Fraida Fund

This reference sheet summarizes some (not all!) of the *data manipulation* tasks you'll do most often throughout the course.

## Numpy arrays: format + shape

Numeric data is often stored as a numpy array:

- **1D array:** `x.shape == (n,)` (a vector)
- **2D array:** `X.shape == (n_samples, n_features)` (a matrix)
- **Higher-D array** (common for image data): `I.shape == (height, width, channels)` (one image) or `I.shape == (n_images, height, width)` / `I.shape == (n_images, height, width, channels)` (a batch)

Useful attributes:

```
X.shape    # tuple, e.g. (100, 5)
X.ndim     # number of dimensions, e.g. 2
X.dtype    # element type, e.g. float64
```

Axis convention (for 2D X):

- `axis=0`: down the rows (compute one value per **column/feature**)
- `axis=1`: across the columns (compute one value per **row/sample**)

## Pandas: DataFrame + Series, named columns + named index

pandas wraps tabular data with labels:

- **DataFrame:** 2D table with *named columns* and a *named index* (row labels)
- **Series:** 1D labeled array (a single column)

and also allows different columns to have different types.

Useful attributes:

```
df.shape    # (n_rows, n_cols)
df.columns  # column names
df.index    # row labels (may be a RangeIndex, dates, IDs, ...)
df.dtypes   # column data types
```

Two common row selectors:

- `df.loc[...]` uses **labels** (index values)
- `df.iloc[...]` uses **integer positions**

## Common operations (numpy vs pandas)

Assume:

```
import numpy as np
import pandas as pd

X = np.array([[1, 10.0],
              [2, 20.0],
              [3, np.nan]])

df = pd.DataFrame({"a": [1, 2, 3],
                  "g": ["A", "A", "B"],
                  "b": [10.0, 20.0, np.nan]},
                  index=["r1", "r2", "r3"])
```

Basic operation (what + why)	How in numpy (example)	How in pandas (example)
<b>Inspect size/shape</b> (sanity-check what you loaded)	<code>X.shape</code> → (3, 2)	<code>df.shape</code> → (3, 2)
<b>Inspect "labels"</b> (know what columns/rows mean)	(no built-in labels)	<code>df.columns</code> , <code>df.index</code>
<b>Select a column</b> (work with one feature)	<code>x = X[:, 0]</code> → 1D array	<code>s = df["a"]</code> → Series
<b>Select multiple columns</b> (subset features)	<code>X2 = X[:, [0, 1]]</code> → 2D array	<code>df2 = df[["a", "b"]]</code> → DataFrame
<b>Select rows by named index</b> (use meaningful row labels/IDs)	(no built-in labels)	<code>row = df.loc["r2"]</code>
<b>Select rows by position</b> (grab a specific sample)	<code>row = X[1]</code>	<code>row = df.iloc[1]</code>
<b>Filter rows by condition</b> (keep only rows meeting a rule)	<code>mask = X[:, 0] &gt; 1</code> <code>Xpos = X[mask]</code>	<code>dfpos = df[df["a"] &gt; 1]</code>
<b>Sort by a column</b> (e.g. to rank)	<code>idx = np.argsort(X[:, 0])</code> <code>Xs = X[idx]</code>	<code>dfs = df.sort_values(by="a")</code>
<b>Sort while keeping correspondence</b> (sort X and y together)	<code>idx = np.argsort(X[:, 0])</code> <code>Xs = X[idx]</code> <code>ys = y[idx]</code>	<code>dfs = df.sort_values("a")</code> <code>ys = y.loc[dfs.index]</code>
<b>Conditionally assign values</b> (create/update a column using a rule)	<code>X2 = np.where(X &gt; 0, X, 0)</code>	<code>df["b2"] = df["b"].where(df["b"] &gt; 0, 0)</code>
<b>Conditionally get indices</b> (find which rows match a rule)	<code>idx = np.where(X[:, 0] &gt; 1)[0]</code> → row indices	<code>idx = df.index[df["a"] &gt; 1]</code> → index labels
<b>Compute summary statistics</b> (describe features)	<code>m = np.mean(X, axis=0)</code>	<code>m = df.mean(numeric_only=True)</code>
<b>Argmax/argmin</b> (index of max/min; e.g. "best/worst" row)	<code>i_max = np.argmax(X[:, 0])</code> <code>i_min = np.argmin(X[:, 0])</code>	<code>i_max = df["a"].idxmax()</code> <code>i_min = df["a"].idxmin()</code>
<b>Group + aggregate</b> (summarize by category)	(not a core numpy pattern)	<code>means = df.groupby("g")["b"].mean()</code>
<b>Stack/concat columns</b> (combine features)	<code>X3 = np.column_stack([X1, X2])</code>	<code>df3 = pd.concat([df1, df2], axis=1)</code>

Basic operation (what + why)	How in numpy (example)	How in pandas (example)
<b>Reshape 1D to 2D</b> (match API expectations like (n_rows, 1))	<pre>x_col = x.reshape(-1, 1) → shape (n_rows, 1) x_row = x.reshape(1, -1) → shape (1, n_rows) x_1d = X.reshape(-1,) → shape (n_rows*n_cols,)</pre>	<pre>s = df["a"] → Series (1D) df[["a"]] → DataFrame (2D)</pre>
<b>Create a new feature</b> (feature engineering)	<pre>new_feature = X[:,0] * X[:,1] Xnew = np.column_stack([X, new_feature])</pre>	<pre>df = df.assign(new_feature=df["a"] * df["b"])</pre>
<b>Ordinal-encode categories</b> (preserve order info)	<pre>map_ = {"low":1, "med":2, "high":3} x = np.array([map_[v] for v in vals])</pre> <p>(not a core numpy pattern)</p>	<pre>map_ = {"low":1, "med":2, "high":3} s = df["level"].map(map_)</pre>
<b>One-hot encode categories</b> (no implied ordering)		<pre>df_ohe = pd.get_dummies(df, columns=["level"], dtype=int)</pre>
<b>Read from a file</b> (load data)	<pre>X = np.load("X.npy") or np.loadtxt for text</pre>	<pre>df = pd.read_csv("data.csv") common: sep, header, index_col</pre>
<b>Convert between numpy ↔ pandas</b> (use the right tool)	<pre>df = pd.DataFrame(X, columns=["a", "b"])</pre>	<pre>X = df.to_numpy()</pre>