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:

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

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