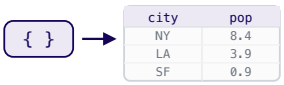
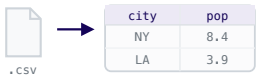
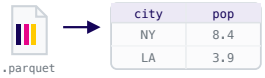
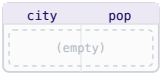
pandas

The everyday pandas moves, from building and cleaning frames to grouping, reshaping, and reading or writing them.

Build a Frame

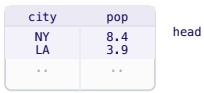
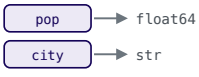
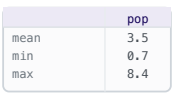
From dicts, lists, or files into a DataFrame.

PURPOSE	COMMAND	RESULT
From a dict of columns.	`pd.DataFrame({"city": cities, "pop": pops})`	
One labeled column (Series).	`pd.Series([8.4, 3.9, 0.9], name="pop")`	
From row records.	`pd.DataFrame.from_records(rows)`	
Read a CSV file.	`pd.read_csv("cities.csv")`	
Read Parquet (columnar).	`pd.read_parquet("cities.parquet")`	
Empty frame with set columns.	`pd.DataFrame(columns=["city", "pop"])`	

a DataFrame is a dict of aligned, named columns (Series) sharing one index

Look at It

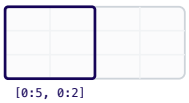
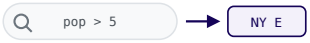
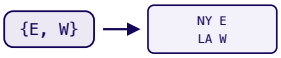
Shape, types, summary, a peek at the data.

PURPOSE	COMMAND	RESULT
First / last n rows.	`df.head(5)` · `df.tail(5)`	
Rows and columns count.	`df.shape`	
Schema, dtypes, non-null.	`df.info()`	
Per-column dtypes.	`df.dtypes`	
Numeric summary stats.	`df.describe()`	
Count distinct values.	`df["region"].value_counts()`	

look before you transform: shape, dtypes, and where the nulls hide

Pick Rows and Columns

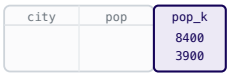
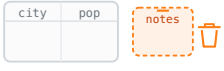
Labels with .loc, positions with .iloc, masks for filtering.

PURPOSE	COMMAND	RESULT
Select columns by name.	`df[["city", "pop"]]`	
Select by label (.loc).	`df.loc[df["pop"] > 5, "city"]`	
Select by integer position.	`df.iloc[0:5, 0:2]`	
Filter with a boolean mask.	`df[df["pop"] > 5]`	
Filter with a query string.	`df.query("pop > 5 and region == 'E'")`	
Keep rows where value in set.	`df[df["region"].isin(["E", "W"])]`	

.loc takes labels and masks; .iloc takes integer positions

Make New Columns

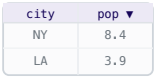
Add, rename, retype, and transform values.

PURPOSE	COMMAND	RESULT
Add or replace columns.	`df.assign(pop_k=df["pop"] * 1000)`	
Rename columns.	`df.rename(columns={"pop": "population"})`	
Change a column's dtype.	`df.astype({"pop": "int64"})`	
Elementwise string transform.	`df["city"].str.upper()`	
Map / apply a function.	`df["pop"].apply(round)`	
Drop columns.	`df.drop(columns=["notes"])`	

most calls return a new frame; chain them rather than mutating in place

Fix Missing & Messy Data

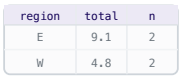
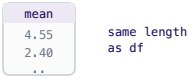
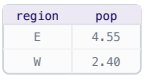
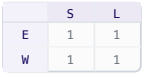
Nulls, duplicates, and ordering.

PURPOSE	COMMAND	RESULT
Count nulls per column.	`df.isna().sum()`	
Drop rows with any null.	`df.dropna()`	
Fill nulls with a value.	`df.fillna(0)`	
Forward-fill nulls.	`df.ffill()`	
Remove duplicate rows.	`df.drop_duplicates()`	
Sort by column values.	`df.sort_values("pop", ascending=False)`	

isna().sum() finds nulls; dropna or fillna/ffill resolve them

Group and Summarize

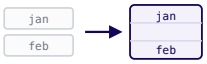
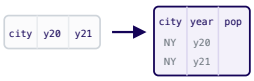
Split into groups, aggregate, summarize into tables.

PURPOSE	COMMAND	RESULT
Group then aggregate.	`df.groupby("region")["pop"].sum()`	
Named aggregations.	`df.groupby("region").agg(` `  total=("pop","sum"), n=("pop","size"))`	
Count rows per group.	`df.groupby("region").size()`	
Broadcast a group stat back.	`df.groupby("region")["pop"]` `  .transform("mean")`	
Pivot to a summary table.	`df.pivot_table(index="region",` `  values="pop", aggfunc="mean")`	
Frequency cross-tab.	`pd.crosstab(df["region"], df["size"])`	

split, apply, combine: the group key lands in the result's index

Reshape and Combine

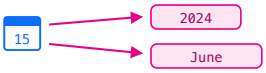
Join frames, stack rows, wide-to-long and back.

PURPOSE	COMMAND	RESULT
Join on a key column.	`pd.merge(left, right, on="id", how="left")`	
Stack rows (same columns).	`pd.concat([df_jan, df_feb])`	
Glue columns side by side.	`pd.concat([df_a, df_b], axis=1)`	
Wide to long (unpivot).	`df.melt(id_vars="city",` `  var_name="year", value_name="pop")`	
Long to wide (pivot).	`df.pivot(index="city",` `  columns="year", values="pop")`	
Expand list-valued cells.	`df.explode("tags")`	

merge joins on a key; melt and pivot are wide/long inverses

Read, Write & Dates

File I/O round-trips and datetime handling.

PURPOSE	COMMAND	RESULT
Write a CSV (no index col).	`df.to_csv("out.csv", index=False)`	
Write Parquet (typed, fast).	`df.to_parquet("out.parquet")`	
Parse strings to datetimes.	`pd.to_datetime(df["date"])`	
Pull out date parts.	`df["date"].dt.year · .dt.month_name()`	
Build a date range index.	`pd.date_range("2024-01-01",` `  periods=12, freq="ME")`	
Downsample a time series.	`df.resample("ME").sum()`	

once a column is real datetimes, .dt and resample unlock calendar work