PYTHON CHEATSHEET

polars

A quick reference for Polars, the fast DataFrame library with a lazy engine, built around expressions for selecting, filtering, grouping, and joining.

Read & Write Data

Eagerly with read_*, or lazily and out-of-core with scan_*.

PURPOSE	COMMAND	RESULT
Read a CSV into memory.	`pl.read_csv("data.csv")`	
Read a Parquet file.	`pl.read_parquet("data.parquet")`	
Scan lazily (no data read).	`pl.scan_parquet("data.parquet")`	
Build a frame from a dict.	`pl.from_dict({"a": [1, 2], "b": [3, 4]})`	
Write to Parquet.	`df.write_parquet("out.parquet")`	
Stream a query to disk.	`lf.sink_parquet("out.parquet")`	

scan_* defers the read so polars can push filters into the file scan

Inspect a Frame

Look before you leap: shape, schema, a peek, and summary stats.

PURPOSE	COMMAND	RESULT
Rows and columns.	`df.shape`	`(1000, 5)`
First few rows.	`df.head(5)`	a v 1 10 2 20 .. ..
Column names and types.	`df.schema`	a → Int64 v → Float64 b → String
Wide, transposed preview.	`df.glimpse()`	a<i64> 1, 2, 3 b<str> "x", "y" v<f64> 1.5, 2.0
Summary statistics.	`df.describe()`	v count 1000 mean 25.0 max 99.0
Nulls per column.	`df.null_count()`	a 0 b 0 v 3

glimpse is the wide transposed peek; describe and null_count flag cleanup work

Select & Add Columns

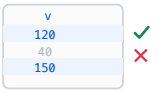
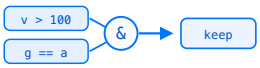
Expressions are the heart of polars: select to keep, with_columns to add.

PURPOSE	COMMAND	RESULT
Keep specific columns.	`df.select("a", "b")`	
Build an expression.	`pl.col("v") * 2`	
Add / overwrite a column.	`df.with_columns((pl.col("v") * 2).alias("v2"))`	
Conditional column.	`pl.when(pl.col("v") > 0)` `  .then(pl.lit("pos")).otherwise(pl.lit("neg"))`	
Change a column's type.	`pl.col("a").cast(pl.Float64)`	
Select by selector.	`df.select(cs.numeric())`	

pl.col("v") * 2 is a recipe, not a value; verbs run it against the frame

Filter & Sort Rows

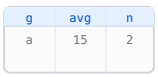
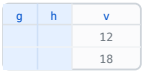
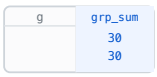
Pick the rows you want with boolean expressions, then order them.

PURPOSE	COMMAND	RESULT
Keep rows matching a test.	`df.filter(pl.col("v") > 100)`	
Combine conditions.	`df.filter((pl.col("v") > 100)` `  & (pl.col("g") == "a"))`	
Keep rows in a set.	`df.filter(pl.col("g").is_in(["a", "b"]))`	
Sort by a column.	`df.sort("v", descending=True)`	
Sort by several keys.	`df.sort(["g", "v"], descending=[False, True])`	
Drop duplicate rows.	`df.unique(subset=["g"])`	

combine conditions with & | ~ , each side wrapped in parentheses

Group & Aggregate

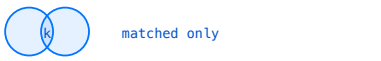
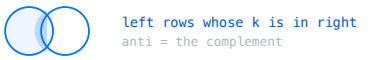
Split by key, aggregate each group, or compute window stats with over.

PURPOSE	COMMAND	RESULT
Sum within each group.	`df.group_by("g").agg(pl.col("v").sum())`	
Count rows per group.	`df.group_by("g").len()`	
Several aggregates at once.	`df.group_by("g").agg(` `  pl.col("v").mean().alias("avg"), pl.len().alias("n"))`	
Group by multiple keys.	`df.group_by(["g", "h"]).agg(pl.col("v").sum())`	
Stat without collapsing rows.	`df.with_columns(pl.col("v").sum()` `  .over("g").alias("grp_sum"))`	
Time-windowed groups.	`df.group_by_dynamic("t", every="1h")` `  .agg(pl.col("v").mean())`	

over("g") computes a group stat but keeps every original row

Join & Concatenate

Combine frames side-by-side on keys, or stack them.

PURPOSE	COMMAND	RESULT
Inner join on a key.	`left.join(right, on="k", how="inner")`	
Keep all left rows.	`left.join(right, on="k", how="left")`	
Outer / full join.	`left.join(right, on="k", how="full")`	
Filtering joins (semi / anti).	`left.join(right, on="k", how="semi")`	
Stack rows (vertical).	`pl.concat([df1, df2])`	
Stack columns (horizontal).	`pl.concat([df1, df2], how="horizontal")`	

joins combine by key; concat just stacks frames by rows or columns

Reshape & Restructure

Pivot wide, unpivot long, explode lists, and rename.

PURPOSE	COMMAND	RESULT
Wide to long (unpivot).	`df.unpivot(index="id", on=["x", "y"])`	
Long to wide (pivot).	`df.pivot("var", index="id", values="val")`	
One list to many rows.	`df.explode("items")`	
Rename columns.	`df.rename({"x": "X"})`	
Drop columns.	`df.drop("tmp")`	
Flip rows and columns.	`df.transpose()`	

unpivot gathers value columns into rows; pivot spreads them back out

Clean: Strings, Dates & Nulls

The everyday tidy-up: namespaced string and date ops, plus null handling.

PURPOSE	COMMAND	RESULT
Transform text.	`pl.col("s").str.to_uppercase()`	
Match a substring.	`pl.col("s").str.contains("ell")`	
Parse text to a date.	`pl.col("d").str.to_date()`	
Extract a date part.	`pl.col("d").dt.year()`	
Fill missing values.	`pl.col("v").fill_null(0)`	
Drop rows with nulls.	`df.drop_nulls()`	

.str and .dt namespaces hold type-specific ops; null is explicit, not NaN