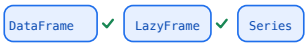
narwhals

narwhals in brief, writing dataframe code once and running it on pandas, Polars, PyArrow, and more.

Wrap & Unwrap

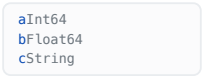
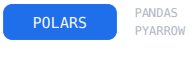
The bridge: turn any native frame into a narwhals frame and back.

PURPOSE	COMMAND	RESULT
Wrap a native frame.	`nw.from_native(df)`	
Restrict to eager frames.	`nw.from_native(df, eager_only=True)`	
Allow a Series through too.	`nw.from_native(s, allow_series=True)`	
Hand the result back.	`nw.to_native(df)`	
Build a frame from scratch.	`nw.from_dict(data, backend="polars")`	

narwhals borrows your frame; to_native returns the exact same native type

Inspect & Metadata

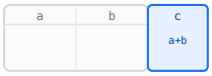
Look at a frame the same way no matter what backend it wraps.

PURPOSE	COMMAND	RESULT
Get (rows, cols).	`df.shape`	
List the column names.	`df.columns`	
Name to dtype mapping.	`df.schema`	
Which engine is underneath?	`df.implementation`	
Reach the native module.	`nw.get_native_namespace(df)`	
Convert to a concrete backend.	`df.to_pandas() / .to_polars()`	

every frame answers shape/columns/schema the same, whatever is underneath

Select, Columns & Expressions

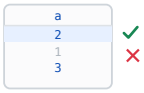
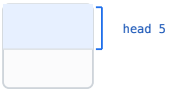
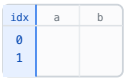
Build columns with nw.col(...) expressions, Polars-style.

PURPOSE	COMMAND	RESULT
Pick columns.	`df.select(nw.col("a", "b"))`	
Add / overwrite columns.	`df.with_columns(` `  (nw.col("a") + nw.col("b")).alias("c"))`	
Reference a literal value.	`nw.lit(0)`	
Conditional column.	`nw.when(nw.col("a") > 1).then(..)` `  .otherwise(..)`	
Combine columns horizontally.	`nw.sum_horizontal("a", "b")`	
Pick columns by selector.	`df.select(ncs.numeric())`	

nw.col(...) expressions are reusable recipes, run by select / with_columns

Filter, Sort & Transform Rows

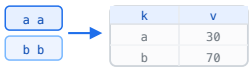
Row-level operations: keep, order, dedupe, reshape.

PURPOSE	COMMAND	RESULT
Keep rows matching a test.	`df.filter(nw.col("a") > 1)`	
Sort by one or more columns.	`df.sort("a", descending=True)`	
First / last N rows.	`df.head(5) / .tail(5)`	
Drop duplicate rows.	`df.unique(subset=["a"])`	
Drop missing values.	`df.drop_nulls()`	
Add a row index.	`df.with_row_index()`	

these verbs read the same on a huge Polars LazyFrame and a tiny pandas frame

Group, Aggregate & Join

Split-apply-combine and relational joins, backend-agnostic.

PURPOSE	COMMAND	RESULT
Group then aggregate.	`df.group_by("k").agg(nw.col("v").sum())`	
Multiple aggregations.	`.agg(nw.col("v").mean(),nw.len())`	
Inner / left / outer join.	`df.join(other, on="id", how="left")`	
Cross join.	`df.join(other, how="cross")`	
Stack frames vertically.	`nw.concat([df1, df2], how="vertical")`	
Window expression over groups.	`nw.col("v").sum().over("k")`	

narwhals maps each verb to the backend's own optimized implementation

Expression Namespaces

Typed sub-APIs (str / dt / cat) that work the same on every backend.

PURPOSE	COMMAND	RESULT
Uppercase strings.	`nw.col("name").str.to_uppercase()`	
Test & slice text.	`nw.col("name").str.contains("a")`	
Extract date parts.	`nw.col("ts").dt.year()`	
Truncate datetimes.	`nw.col("ts").dt.truncate("1d")`	
Categorical to categories.	`nw.col("c").cat.get_categories()`	
Parse strings to dates.	`nw.col("d").str.to_datetime(format="%Y-%m-%d")`	

.str .dt .cat .list .struct make typed work uniform across engines

Lazy Frames & I/O

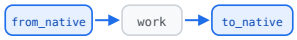
Defer work with LazyFrame; read and write without committing to one engine.

PURPOSE	COMMAND	RESULT
Go lazy (defer execution).	`df.lazy()`	
Trigger the computation.	`lf.collect()`	
Scan a CSV lazily.	`nw.scan_csv("data.csv", backend="polars")`	
Read a CSV eagerly.	`nw.read_csv("data.csv", backend="pandas")`	
Stream a lazy result to disk.	`lf.sink_parquet("out.parquet")`	
Write a frame out.	`df.write_parquet("out.parquet")`	

read_* and scan_* require backend= ; narwhals will not guess the engine

Write Portable Functions

The payoff: one function, every dataframe library.

PURPOSE	COMMAND	RESULT
Auto-wrap/unwrap a function.	`@nw.narwhalify def f(df): ...`	
Write the body once.	`returndf.with_columns(nw.col("a").mean())`	
Manual wrap form.	`df =nw.from_native(df); ... returnnw.to_native(df)`	
Use the stable API.	`importnarwhals.stable.v1as nw`	
Pass native through untouched.	`nw.from_native(x, pass_through=True)`	
Convert a Series the same way.	`nw.from_native(s, series_only=True)`	

@nw.narwhalify accepts and returns whatever dataframe library the caller uses