Polars Cheatsheet

Basic Concepts

Polars is a blazingly fast DataFrame library implemented in Rust, designed for high performance and efficient parallel processing.

- Import: `import polars as pl`.
- Eager Execution: `df = pl.DataFrame(...)` (Executes immediately).
- Lazy Execution: `df_lazy = pl.scan_csv(...)` (Creates an execution plan; executes later with `.collect()`).

Selection and Filtering

- Select Columns: `df.select([pl.col("name"), pl.col("age")])`.
- Filter Rows: `df.filter(pl.col("age") > 25)`.
- Add/Transform Column: `df.with_columns([(pl.col("age") * 2).alias("double_age")])`.

Aggregation and Grouping

```
df.group_by("category").agg([

pl.col("score").mean().alias("avg_score"),

pl.col("score").sum().alias("total_score"),
    pl.count()
])
```

Joining DataTasks

- `df1.join(df2, on="id", how="inner")`: Joins include `inner`, `left`, `outer`, `semi`, `anti`.
- `pl.concat([df1, df2])`: Combine DataFrames vertically or horizontally.

File I/O (Eager vs Lazy)

- Read CSV: `pl.read_csv("data.csv")` (Eager) / `pl.scan_csv("data.csv")` (Lazy).
- Write CSV: `df.write_csv("output.csv")`.
- Read Parquet: `pl.read_parquet("data.parquet")`.

Contexts and Expressions (Key Logic)

Polars uses a powerful expression language.

- `select`: Work on columns.
- `with_columns`: Add or change columns.
- `filter`: Pick rows.
- `group_by`: Aggregate by groups.

Handling Dates and Times

```
df.with_columns([
    pl.col("date").str.to_datetime("%Y-
%m-%d"),

pl.col("date").dt.year().alias("year")
])
```

Missing Data handling

- `df.null_count()`: Check for nulls.
- `df.drop_nulls()`: Remove nulls.
- `df.fill_null(value)`: Fill nulls.

Why Polars? (Pro Tips)

Multithreading

Polars is designed from the ground up to be multithreaded. Expressions are executed in parallel on all available cores.

Memory Efficiency

Uses Apache Arrow as its internal data format, ensuring memory-efficient data representation and zero-copy interoperability.

Lazy API (Query Optimization)

Using `lazy()` allows Polars to perform query optimization (like predicate pushdown and projection pushdown) before actually processing data, potentially saving massive amounts of time and RAM.

- Always use `LazyFrame` for large datasets: `lazy_df.filter(...).select(...).collect()`.