Cheat Sheet - Polars

esuriddick

Introduction

Library

```
import polars as pl
```

DataFrame

```
df = pl.DataFrame({
   "id": [1, 2, 3, 4]
   ,"name": ["Alice", "Bob", "Charlie", "Diana"]
   ,"age": [25, 30, 35, 28]
   ,"score": [85.5, 90.0, 78.3, 92.1]
})
```

LazyFrame

```
lf = pl.LazyFrame({
   "id": [1, 2, 3, 4]
   ,"name": ["Alice", "Bob", "Charlie", "Diana"]
   ,"age": [25, 30, 35, 28]
   ,"score": [85.5, 90.0, 78.3, 92.1]
})
```

Conversion to DataFrame

```
df = lf.collect(engine = EngineType)

pl.Config.set_engine_affinity(EngineType)
```

The values allowed for EngineType are 'in-memory' (default), 'streaming' and 'gpu'.

LazyFrame Schema

```
schema = lf.collect_schema()
names = schema.names()
data_types = schema.dtypes()
n_vars = schema.len()
```

Input/Output

Parquet

```
lf = pl.scan_parquet(
   source = filepath
   ,n_rows = integer / None (default)
)

lf.sink_parquet(
   path = filepath
   ,compression = string (default: 'zstd')
   ,compression_level = int / None (default)
   ,engine = EngineType (default: 'auto')
)
```

CSV

```
lf = pl.scan_csv(
   source = filepath
   ,has_header = bool (default: True)
   ,separator = string (default: ',')
   ,quote_char = string (default: '"')
   ,decimal_comma = bool (default: False)
   ,schema_overrides = dictionary / None (default)
   ,skip_rows = integer (default: 0)
   ,skip_lines = integer (default: 0)
)

lf.sink_csv(
   path = filepath
   ,include_header = bool (default: True)
   ,separator = string (default: ',')
   ,line_terminator = string (default: '\n')
   ,quote_char = string (default: '"')
   ,decimal_comma = bool (default: 'False')
   ,engine = EngineType (default: 'auto')
)
```

Selection

Rows

```
lf.filter(
   (pl.col('age') > 27) | (pl.col('name') == 'Bob')
)

lf.filter(
   (pl.col('age') > 27) & (pl.col('name') == 'Bob')
)

lf.filter(
   pl.col('age') > 27
   ,pl.col('name') == 'Bob'
)
```

Columns

```
lf.select(
   pl.col(['name', 'age', 'score'])
)

Special expressions: pl.all(), pl.exclude('column-
name-01', ..., 'column-name-
n'), pl.col([pl.Int64, pl.Float64, pl.String])
```

Manipulation

Rename Column(s)

```
lf.rename(
   {'age': 'Age'}
)
```

Drop Column(s)

```
lf.drop(
   ['age', 'score']
)
```

Change Data Type

```
lf.cast(
   {'age': pl.Int8}
)
```

Sorting

```
lf.sort(
   ['score', 'age']
   ,descending = bool (default: False)
   ,nulls_last = bool (default: False)
)

lf.select(
   pl.all().sort_by(['score', 'age'])
)

lf.select(
   pl.col(['score', 'age']).set_sorted()
)
```

Index Column

```
lf = lf.with_row_index(name = 'index')
```

Custom Column(s)

```
lf.with_columns(
   age_01 = pl.col('age').round(0)
   ,pl.col('age').round(0).alias('age_02')
   ,pl.col('age').round(0).name.suffix('_03')
   ,pl.col('age').name.prefix('org_')
)
```

Conditional Column(s)

```
lf.select(
   pl.when(
      (pl.col('score') > 80)
      & (pl.col('age') > 30)
   )
   .then(pl.lit('Pass'))
   .when(
      pl.col('score') > 60
   )
   .then(pl.lit('Pending'))
   .otherwise(pl.lit('Fail'))
   .alias('result')
)
```

Functions

User-defined

```
def sqrd(lf: pl.LazyFrame, col_name: str) ->
pl.LazyFrame:
   return (
      lf.with_columns(
         pl.col(col_name) ** 2
      )
   )
lf.pipe(
   sqrd
   ,col_name = 'age'
)
```

Lambda

```
lf.with_columns(
   lf.pipe(
      lambda temp_df: pl.col("age") ** 2
   )
)
```

Statistics

XXX

```
xxx
```

XXX

```
xxx
```