Polars Cheat Sheet

Open in Colab

General

Install

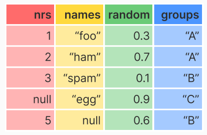
```
pip install polars
```

Import

```
import polars as pl
```

Creating/reading DataFrames

Create DataFrame



nrs	names	random	groups
1	"foo"	0.3	"A"
2	"ham"	0.7	"A"
3	"spam"	0.1	"B"
null	"egg"	0.9	"C"
5	null	0.6	"B"

```
df = pl.DataFrame(
  {
    "nrs": [1, 2, 3, None, 5],
    "names": ["foo", "ham", "spam", "egg", None
    "random": [0.3, 0.7, 0.1, 0.9, 0.6],
    "groups": ["A", "A", "B", "C", "B"],
  }
)
```

Read CSV

```
df = pl.read_csv("https://j.mp/iriscsv",
                 has_header=True)
```

Read parquet

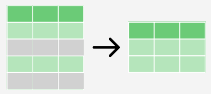
```
df = pl.read_parquet("path.parquet",
                     columns=["select", "column
```

Expressions

Polars expressions can be performed in sequence. This improves readability of code.

```
df \
  .filter(pl.col("nrs") < 4) \
  .groupby("groups") \
  .agg(
    pl \
      .all() \
      .sum()
)
```

Subset Observations - rows



Filter: Extract rows that meet logical criteria.

```
df.filter(pl.col("random") > 0.5)
df.filter(
  (pl.col("groups") == "B")
  & (pl.col("random") > 0.5)
)
```

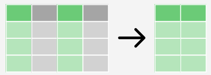
Sample

```
# Randomly select fraction of rows.
df.sample(frac=0.5)

# Randomly select n rows.
df.sample(n=2)
```

Select first and last rows

```
# Select first n rows
df.head(n=2)

# Select last n rows.
df.tail(n=2)
```

Subset Variables - columns



Select multiple columns with specific names

```
df.select(["nrs", "names"])
```

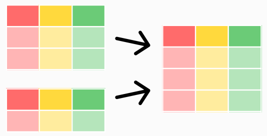
Select columns whose name matches regex

```
df.select(pl.col("^n.*$"))
```

Subsets - rows and columns

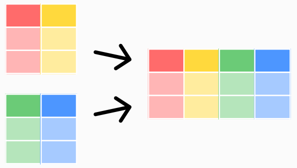
Select rows 2-4

Reshaping Data – Change layout, sorting, renaming



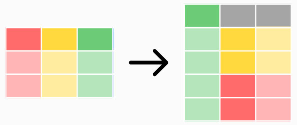
Append rows of DataFrames

```
pl.concat([df, df2])
```



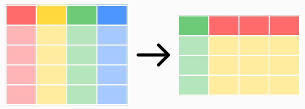
Append columns of DataFrames

```
pl.concat([df, df3], how="horizontal")
```



Gather columns into rows

```
df.melt(
  id_vars="nrs",
  value_vars=["names", "groups"]
)
```



Spread rows into columns

```
df.pivot(values="nrs", index="groups",
         columns="names")
```

Summarize Data

Count number of rows with each unique value of variable

```
df["groups"].value_counts()
```

# of rows in DataFrame

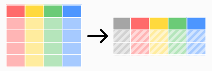
```
len(df)
# or
df.height
```

Tuple of # of rows, # of columns in DataFrame

```
df.shape
```

# of distinct values in a column

```
df["groups"].n_unique()
```



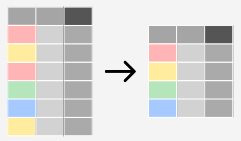
Basic descriptive and statistics for each column

```
df.describe()
```

Aggregation functions

```
df.select(
    [
      # Sum values
      pl.sum("random").alias("sum"),

      # Minimum value
      pl.min("random").alias("min"),

      # Maximum value
      pl.max("random").alias("max"),
      # or
      pl.col("random").max().alias("other_max")

      # Standard deviation
      pl.std("random").alias("std dev"),

      # Variance
      pl.var("random").alias("variance"),

      # Median
      pl.median("random").alias("median"),

      # Mean
      pl.mean("random").alias("mean"),

      # Quantile
      pl.quantile("random", 0.75) \
        .alias("quantile_0.75"),
      # or
      pl.col("random").quantile(0.75) \
        .alias("other_quantile_0.75"),

      # First value
      pl.first("random").alias("first"),
```

Group Data



Group by values in column named "col", returning a GroupBy object

```
df.groupby("groups")
```

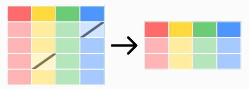
All of the aggregation functions from above can be applied to a group as well

```
df.groupby(by="groups").agg(
  [
      # Sum values
      pl.sum("random").alias("sum"),

      # Minimum value
      pl.min("random").alias("min"),

      # Maximum value
      pl.max("random").alias("max"),
      # or
      pl.col("random").max().alias("other_max")

      # Standard deviation
      pl.std("random").alias("std_dev"),

      # Variance
      pl.var("random").alias("variance"),

      # Median
      pl.median("random").alias("median"),

      # Mean
      pl.mean("random").alias("mean"),

      # Quantile
      pl.quantile("random", 0.75) \
        .alias("quantile_0.75"),
      # or
      pl.col("random").quantile(0.75) \
        .alias("other_quantile_0.75"),

      # First value
      pl.first("random").alias("first"),
  ]
)
```

Additional GroupBy functions

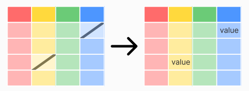
```
df.groupby(by="groups").agg(
  [
    # Count the number of values in each group
    pl.count("random").alias("size"),

    # Sample one element in each group
    pl.col("names").apply(
      lambda group_df: group_df.sample(1)
    ),
```

Handling Missing Data



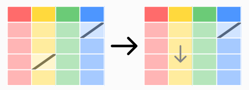
Drop rows with any column having a null value

```
df.drop_nulls()
```



Replace null values with given value

```
df.fill_null(42)
```



Replace null values using forward strategy

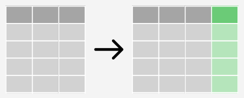
```
df.fill_null(strategy="forward")
```

Other fill strategies are "backward", "min", "max", "mean", "zero" and "one"

Replace floating point NaN values with given value

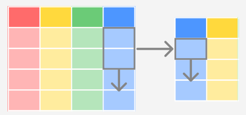
```
df.fill_nan(42)
```

Make New Columns



Add a new columns to the DataFrame

Rolling Functions



The following rolling functions are available

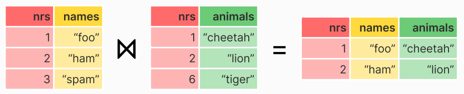
```
df.select(
    [
        # Rolling maximum value
        pl.col("random") \
          .rolling_max(window_size=2) \
          .alias("rolling_max"),

        # Rolling mean value
        pl.col("random") \
          .rolling_mean(window_size=2) \
          .alias("rolling_mean"),

        # Rolling median value
        pl.col("random") \
          .rolling_median(
              window_size=2, min_periods=2) \
          .alias("rolling_median"),

        # Rolling minimum value
        pl.col("random") \
          .rolling_min(window_size=2) \
          .alias("rolling_min"),

        # Rolling standard deviation
        pl.col("random") \
          .rolling_std(window_size=2) \
          .alias("rolling_std"),

        # Rolling sum values
        pl.col("random") \
          .rolling_sum(window_size=2) \
          .alias("rolling_sum"),

        # Rolling variance
        pl.col("random") \
          .rolling_var(window_size=2) \
          .alias("rolling_var"),

        # Rolling quantile
        pl.col("random") \
          .rolling_quantile(
            quantile=0.75, window_size=2,
            min_periods=2
          ) \
          .alias("rolling_quantile"),

        # Rolling skew
        pl.col("random") \
          .rolling_skew(window_size=2) \
          .alias("rolling_skew"),

        # Rolling custom function
        pl.col("random") \
          .rolling_apply(
            function=np.nanstd, window_size=2)
          .alias("rolling_apply"),
    ]
)
```

Window Functions

Window functions allow to group by several columns simultaneously

```
df.select(
    [
        "names",
        "groups",
        pl.col("random").sum().over("names") \
          .alias("sum_by_names"),
        pl.col("random").sum().over("groups") \
          .alias("sum_by_groups"),
    ]
)
```

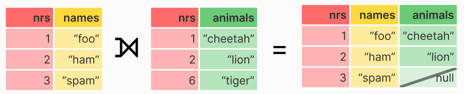
Combine Data Sets



Inner Join

Retains only rows with a match in the other set.

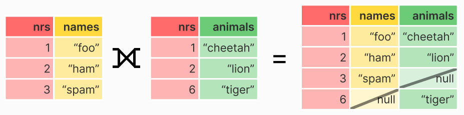
```
df.join(df4, on="nrs")
# or
df.join(df4, on="nrs", how="inner")
```



Left Join

Retains each row from "left" set (df).

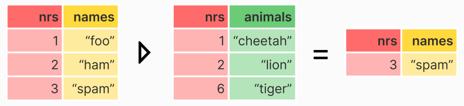
```
df.join(df4, on="nrs", how="left")
```



Outer Join

Retains each row, even if no other matching row exists.

```
df.join(df4, on="nrs", how="outer")
```



Anti Join

Contains all rows from df that do not have a match in df4.

```
df.join(df4, on="nrs", how="anti")
```