polars

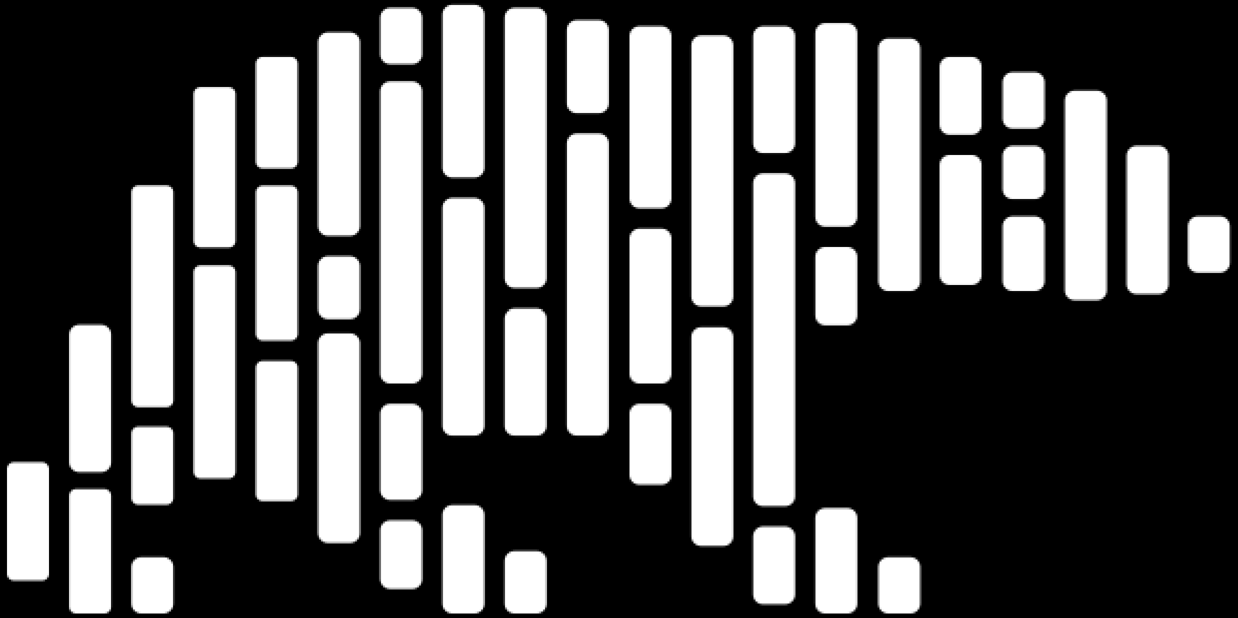
Done by:

Yosef Alsheikh qasem

Intelligence Group

What is polars

Polars is a free tool for working with data, It's one of the fastest ways to process data on a single computer, It has a clear and organized system that makes it simple and efficient to use.



Intelligence Group

Why use polars?

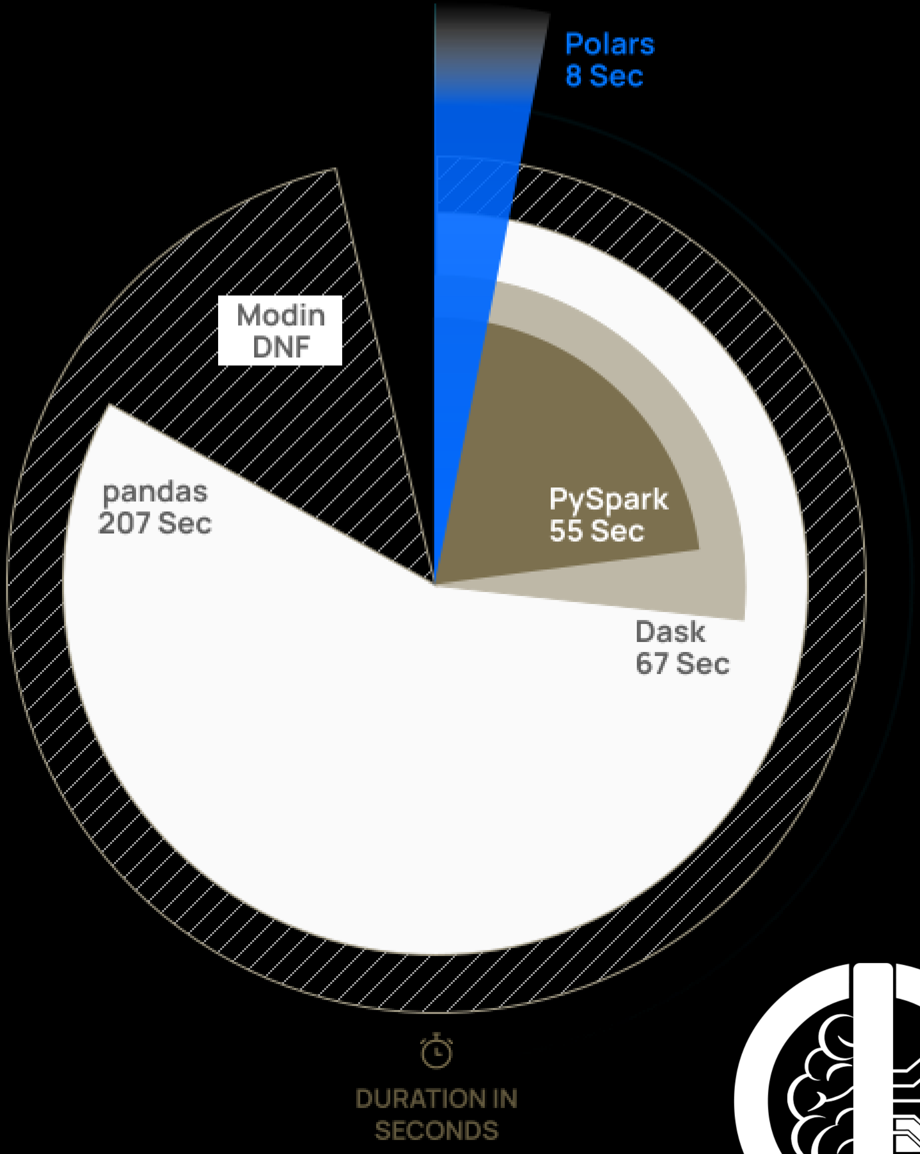
1-Fast

- Polars is built from scratch to be fast and efficient.
- written in Rust for better speed.
- Designed to handle multiple tasks at the same time (parallel processing).
- Works with column-based data and uses modern techniques for high performance.

Intelligence Group

Why use polars?

Tools Compared:
-Pandas: A popular Python library for data analysis and manipulation.
-Modin: A framework that speeds up Pandas by parallelizing its operations.
-PySpark: A Python API for Apache Spark, a distributed computing framework.
-Dask: A flexible parallel computing library in Python.

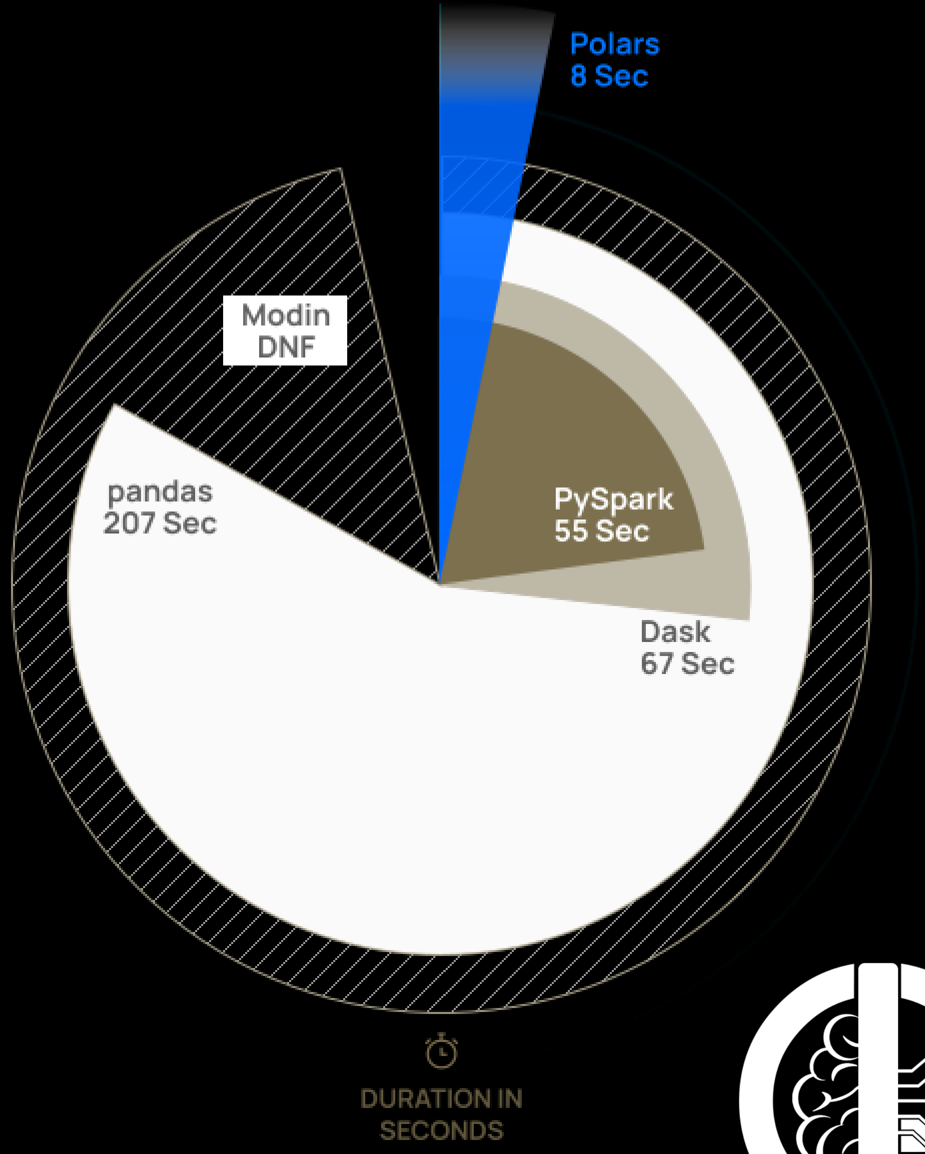


Intelligence Group

Why use polars?

Polars was tested on real-world data tasks and outperformed other tools by using parallel processing, smart algorithms, and modern CPU features. It can be over 30 times faster than Pandas.

Read more:
https://pola.rs/posts/benchmarks/



Intelligence Group

Why use polars?

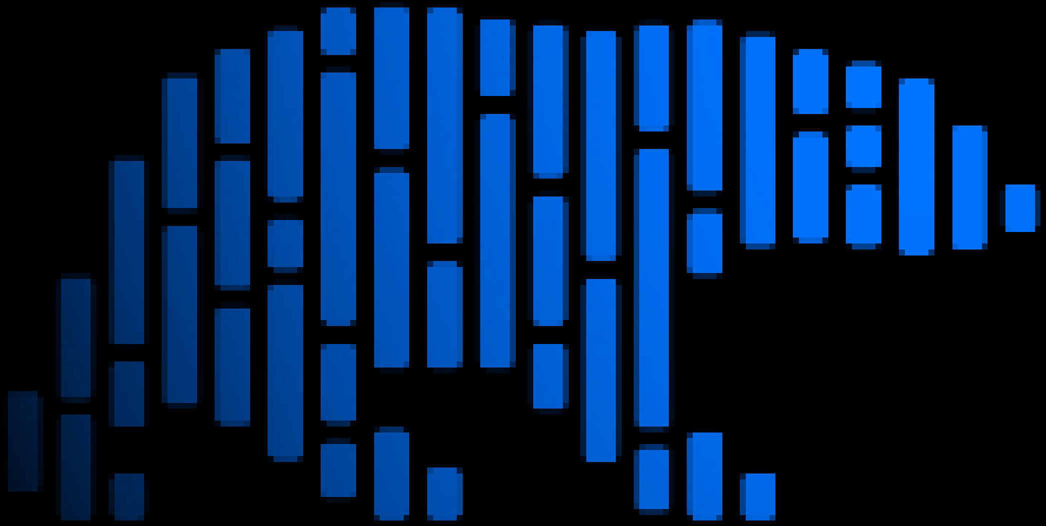
2-Easy to use

Polars is easy to use, with a simple design similar to Pandas, so you can learn it quickly. It has tools to handle data easily and great guides and community support to help you get started.

Why use polars?

3-open source

Polars is and always will be open source, driven by an active community of developers, everyone is encouraged to add new features and contribute.



Intelligence Group

Parallelism in polars

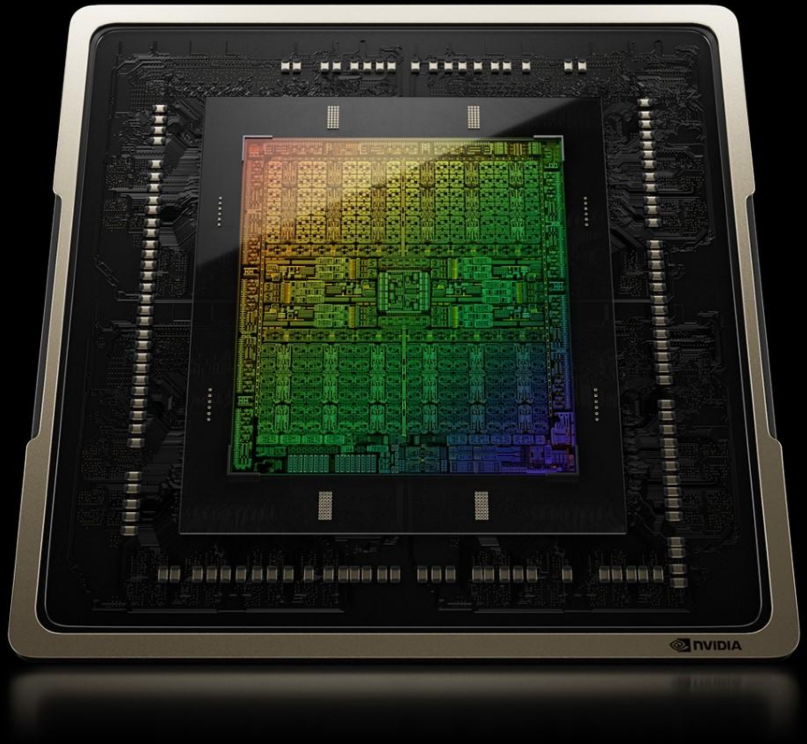
Polars is a high-performance DataFrame library for Python that leverages parallelism at multiple levels to accelerate data processing.

Its efficiency is achieved through a combination of SIMD (Single Instruction, Multiple Data) and multi-threaded parallel programming.

Intelligence Group

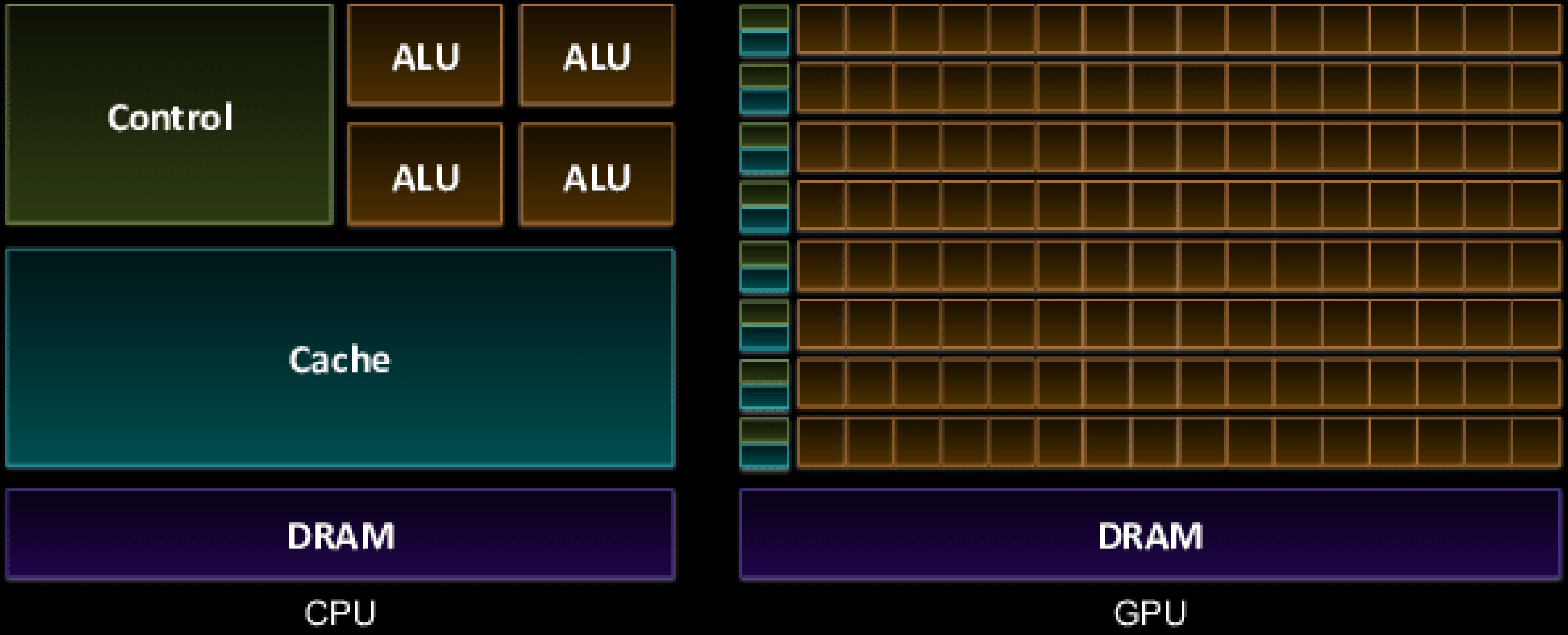
SIMD

SIMD (Single Instruction, Multiple Data) is a computational paradigm used in modern processors to perform the same operation on multiple data points simultaneously. This approach is particularly well-suited for applications that involve repetitive tasks on large datasets, such as graphics processing, scientific simulations, and data analytics.



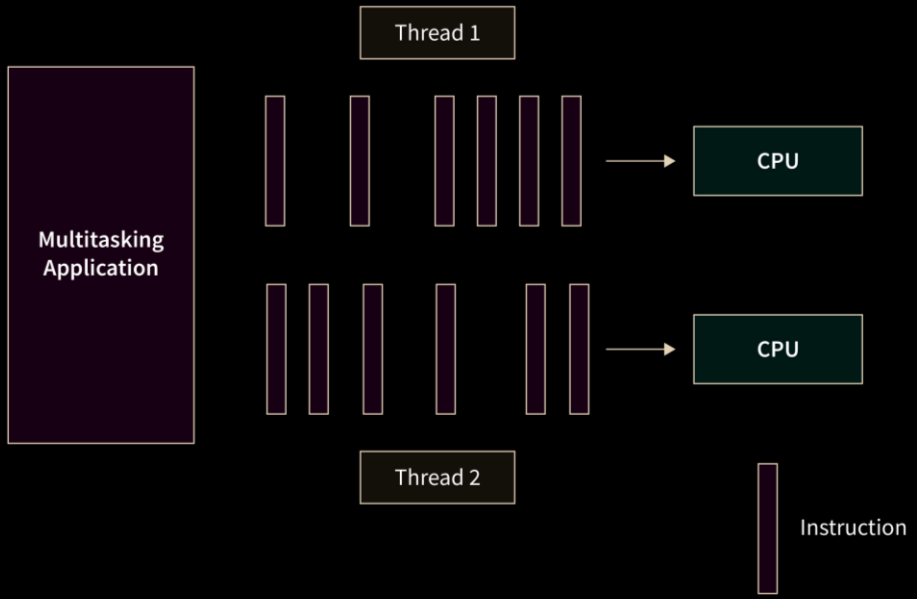
Intelligence Group

SIMD



Parallel programming

Parallel programming is a method of writing software that can execute multiple tasks simultaneously by dividing a problem into smaller sub-problems, which are solved concurrently. This approach maximizes the utilization of computational resources and significantly improves performance, particularly for tasks that are computationally intensive.



Intelligence Group

Data formats support

Polars supports reading and writing to all common data formats. This allows you to easily integrate Polars with your existing data stack.

Data formats support

Text: CSV & JSON

Binary: Parquet, Delta Lake, AVRO & Excel

IPC: Feather, Arrow

Databases: MySQL, Postgres, SQL Server, Sqlite, Redshift & Oracle

Cloud storage: S3, Azure Blob & Azure File

Intelligence Group

Installing polars

To get started with Polars, simply install the library on your device by running the following command in your Python environment:

-pip install polars

Intelligence Group

Opening repository

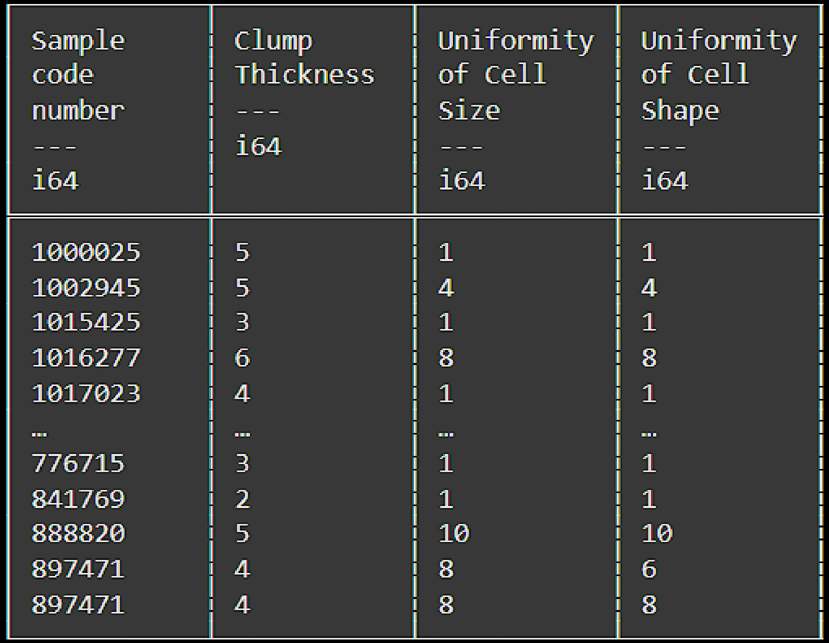
github.com/Yosef024/polars-workshop024

Intelligence Group

Reading data

reading data involves loading datasets from various file formats into a DataFrame, which is the core structure used for data manipulation.

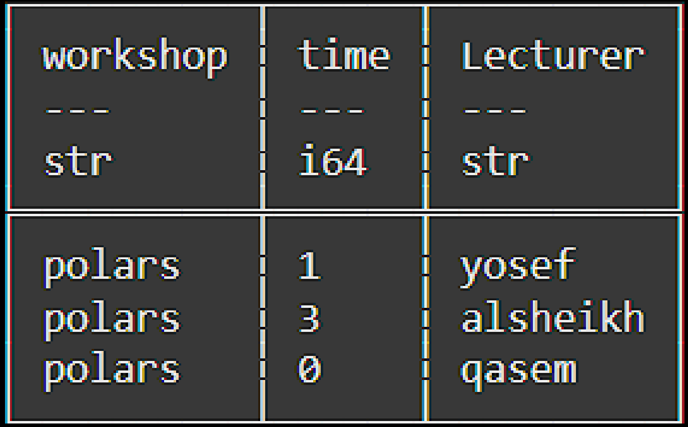
```python
import polars as pl
dataset=pl.read_csv('Data.csv')
```



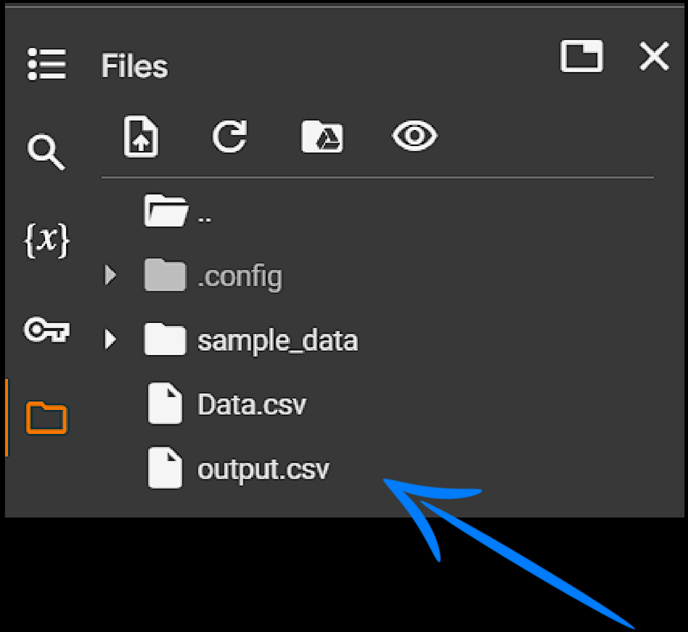
```
┌─────────────┬────────────┬────────────┬────────────┐
│ Sample      ┆ Clump      ┆ Uniformity ┆ Uniformity │
│ code        ┆ Thickness  ┆ of Cell    ┆ of Cell    │
│ number      ┆ ---        ┆ Size       ┆ Shape      │
│ ---         ┆ i64        ┆ ---        ┆ ---        │
│ i64         ┆            ┆ i64        ┆ i64        │
╞═════════════╪════════════╪════════════╪════════════╡
│ 1000025     ┆ 5          ┆ 1          ┆ 1          │
│ 1002945     ┆ 5          ┆ 4          ┆ 4          │
│ 1015425     ┆ 3          ┆ 1          ┆ 1          │
│ 1016277     ┆ 6          ┆ 8          ┆ 8          │
│ 1017023     ┆ 4          ┆ 1          ┆ 1          │
│ …           ┆ …          ┆ …          ┆ …          │
│ 776715      ┆ 3          ┆ 1          ┆ 1          │
│ 841769      ┆ 2          ┆ 1          ┆ 1          │
│ 888820      ┆ 5          ┆ 10         ┆ 10         │
│ 897471      ┆ 4          ┆ 8          ┆ 6          │
│ 897471      ┆ 4          ┆ 8          ┆ 8          │
└─────────────┴────────────┴────────────┴────────────┘
```

Writing data

```
import polars as pl
df = pl.DataFrame({
    "workshop": ["polars", "polars", "polars"],
    "time": [1, 3, 0],
    "Lecturer": ["yosef", "alsheikh", "qasem"]
})
print(df)
df.write_csv("output.csv")
```



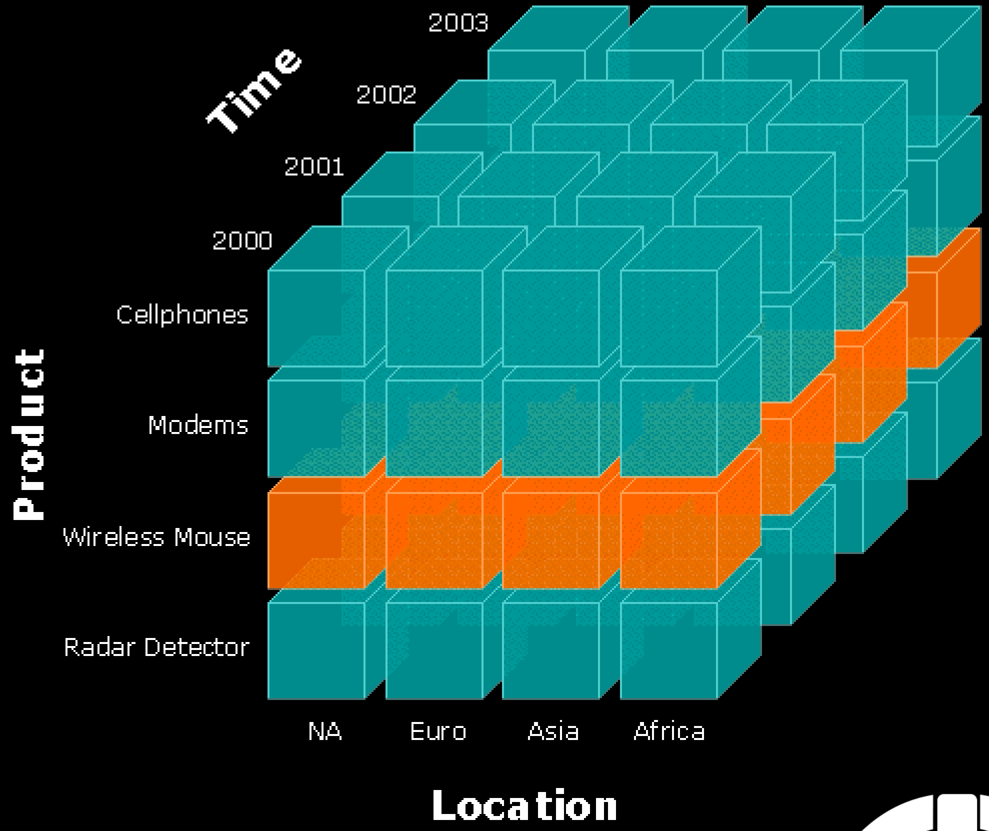
```
┌──────────┬──────┬──────────┐
│ workshop ┆ time ┆ Lecturer │
│ ---      ┆ ---  ┆ ---      │
│ str      ┆ i64  ┆ str      │
╞══════════╪══════╪══════════╡
│ polars   ┆ 1    ┆ yosef    │
│ polars   ┆ 3    ┆ alsheikh │
│ polars   ┆ 0    ┆ qasem    │
└──────────┴──────┴──────────┘
```



Column selection

This function allows you to select specific columns from a DataFrame. You can also perform operations on the selected columns.

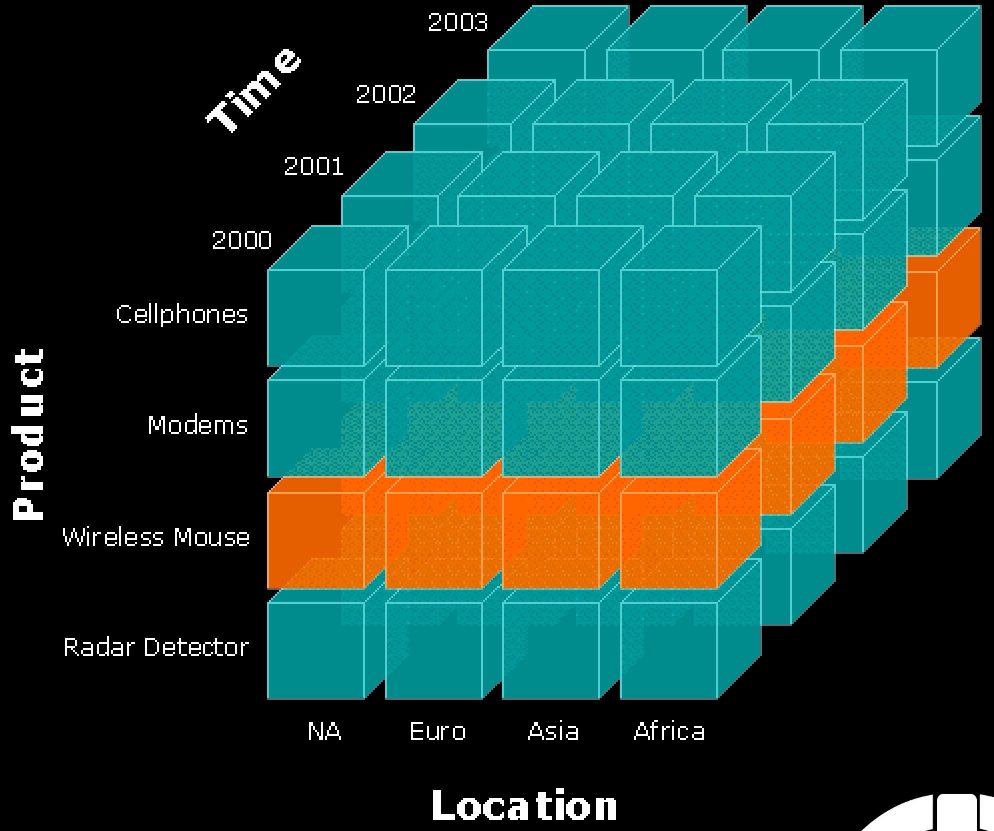
```
dataset.select(["col1", "col2"])
```



Row filtering

Filters rows based on a condition or a set of conditions.

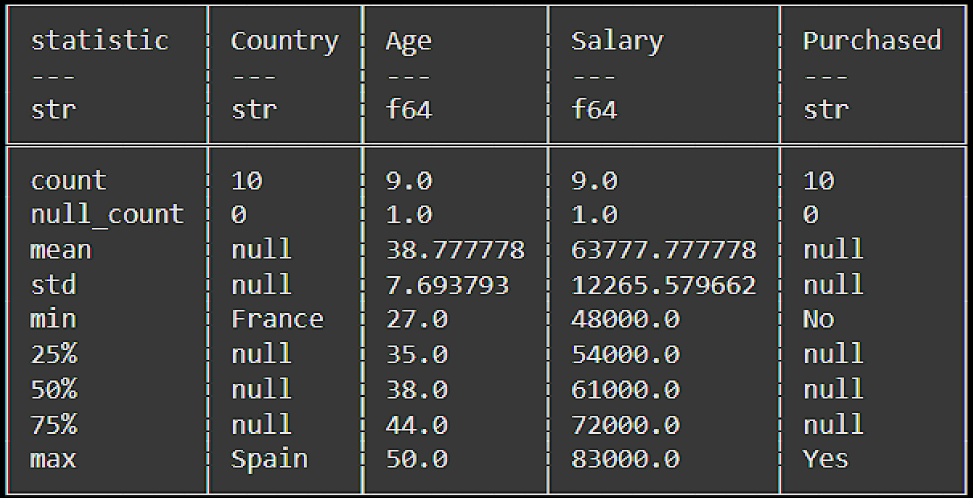
```
dataset.filter(pl.col("age") > 30)
```



describe

The describe() function in Polars provides a quick statistical summary of a DataFrame.

1-Count: Number of non-null values in each column.
2-Mean: Average of numeric values.
3-Std: Standard deviation for numeric values.

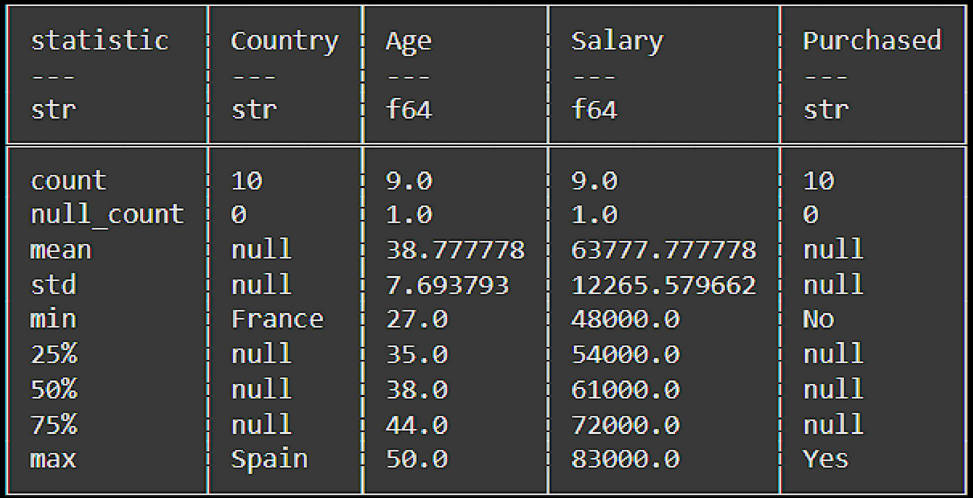


```
statistic   | Country | Age       | Salary       | Purchased
---         | ---     | ---       | ---          | ---
str         | str     | f64       | f64          | str
count       | 10      | 9.0       | 9.0          | 10
null_count  | 0       | 1.0       | 1.0          | 0
mean        | null    | 38.777778 | 63777.777778 | null
std         | null    | 7.693793  | 12265.579662 | null
min         | France  | 27.0      | 48000.0      | No
25%         | null    | 35.0      | 54000.0      | null
50%         | null    | 38.0      | 61000.0      | null
75%         | null    | 44.0      | 72000.0      | null
max         | Spain   | 50.0      | 83000.0      | Yes
```

describe

4-Min/Max: Minimum and maximum values in each column.
5-Median: Middle value of sorted data for numeric columns.
6-25th/75th Percentiles: Values at the 25% and 75% positions.

```
print(dataset.describe())
```



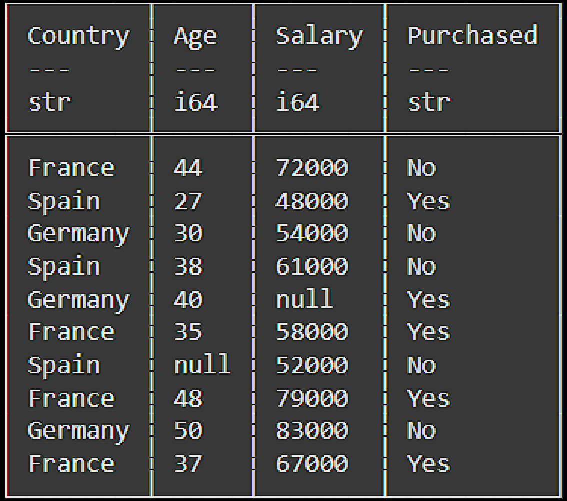
```
┌────────────┬─────────┬───────────┬──────────────┬───────────┐
│ statistic  ┆ Country ┆ Age       ┆ Salary       ┆ Purchased │
│ ---        ┆ ---     ┆ ---       ┆ ---          ┆ ---       │
│ str        ┆ str     ┆ f64       ┆ f64          ┆ str       │
╞════════════╪═════════╪═══════════╪══════════════╪═══════════╡
│ count      ┆ 10      ┆ 9.0       ┆ 9.0          ┆ 10        │
│ null_count ┆ 0       ┆ 1.0       ┆ 1.0          ┆ 0         │
│ mean       ┆ null    ┆ 38.777778 ┆ 63777.777778 ┆ null      │
│ std        ┆ null    ┆ 7.693793  ┆ 12265.579662 ┆ null      │
│ min        ┆ France  ┆ 27.0      ┆ 48000.0      ┆ No        │
│ 25%        ┆ null    ┆ 35.0      ┆ 54000.0      ┆ null      │
│ 50%        ┆ null    ┆ 38.0      ┆ 61000.0      ┆ null      │
│ 75%        ┆ null    ┆ 44.0      ┆ 72000.0      ┆ null      │
│ max        ┆ Spain   ┆ 50.0      ┆ 83000.0      ┆ Yes       │
└────────────┴─────────┴───────────┴──────────────┴───────────┘
```

Intelligence Group

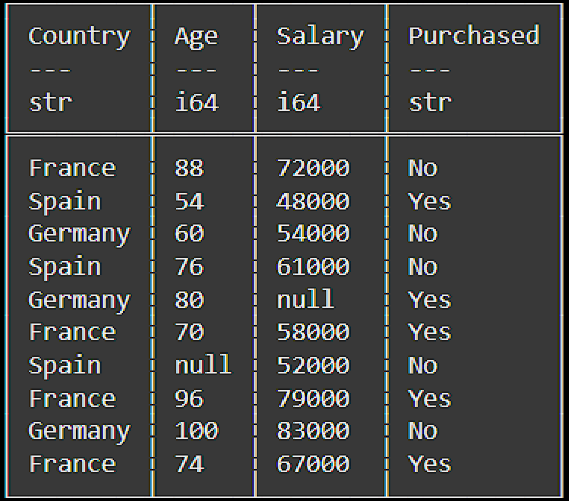
Modifying columns

Adds or modifies columns in the DataFrame.

```
dataset.with_columns([pl.col("age") * 2])
```



```
Country  Age   Salary  Purchased
---      ---   ---     ---
str      i64   i64     str

France   44    72000   No
Spain    27    48000   Yes
Germany  30    54000   No
Spain    38    61000   No
Germany  40    null    Yes
France   35    58000   Yes
Spain    null  52000   No
France   48    79000   Yes
Germany  50    83000   No
France   37    67000   Yes
```



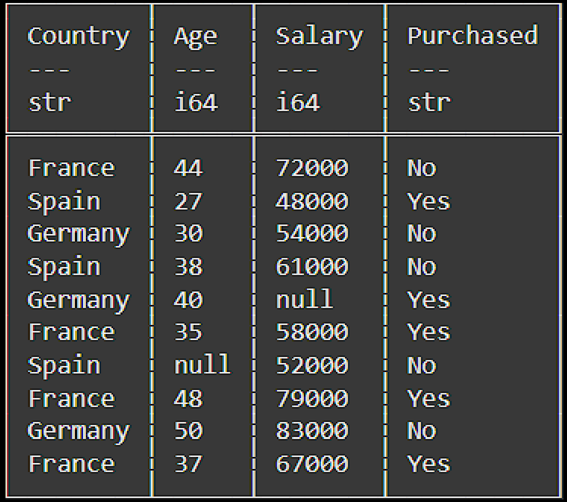
```
Country  Age   Salary  Purchased
---      ---   ---     ---
str      i64   i64     str

France   88    72000   No
Spain    54    48000   Yes
Germany  60    54000   No
Spain    76    61000   No
Germany  80    null    Yes
France   70    58000   Yes
Spain    null  52000   No
France   96    79000   Yes
Germany  100   83000   No
France   74    67000   Yes
```

Intelligence Group

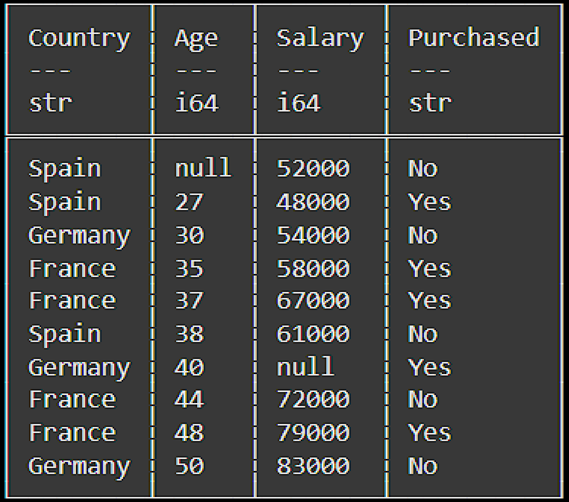
sorting

Sorts the rows in a DataFrame based on one or more columns, either in ascending or descending order.

```
dataset=dataset.sort("Age",descending=False)
```



Country --- str	Age --- i64	Salary --- i64	Purchased --- str
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	38	61000	No
Germany	40	null	Yes
France	35	58000	Yes
Spain	null	52000	No
France	48	79000	Yes
Germany	50	83000	No
France	37	67000	Yes

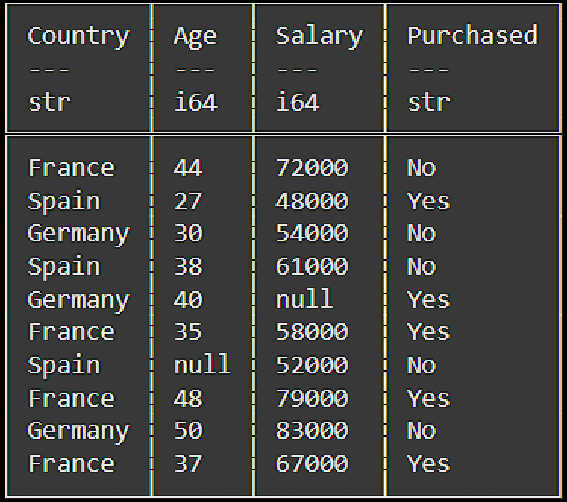


Country --- str	Age --- i64	Salary --- i64	Purchased --- str
Spain	null	52000	No
Spain	27	48000	Yes
Germany	30	54000	No
France	35	58000	Yes
France	37	67000	Yes
Spain	38	61000	No
Germany	40	null	Yes
France	44	72000	No
France	48	79000	Yes
Germany	50	83000	No

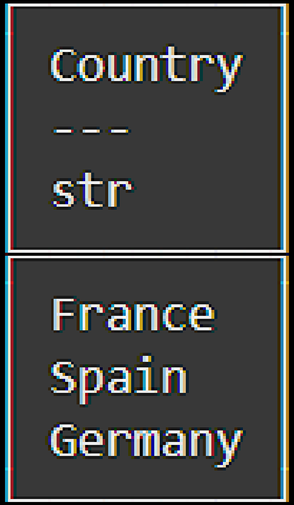
Unique

Returns the unique rows or values from a column, similar to removing duplicates.

```
dataset.select("city").unique()
```



```
Country  | Age  | Salary | Purchased
---      | ---  | ---    | ---
str      | i64  | i64    | str

France   | 44   | 72000  | No
Spain    | 27   | 48000  | Yes
Germany  | 30   | 54000  | No
Spain    | 38   | 61000  | No
Germany  | 40   | null   | Yes
France   | 35   | 58000  | Yes
Spain    | null | 52000  | No
France   | 48   | 79000  | Yes
Germany  | 50   | 83000  | No
France   | 37   | 67000  | Yes
```

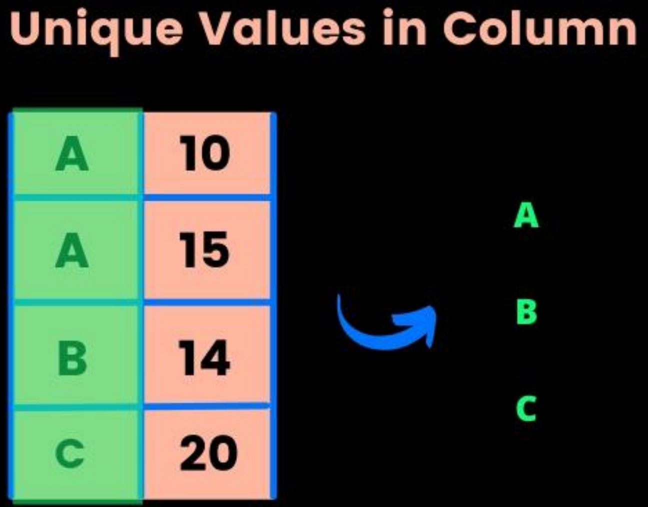


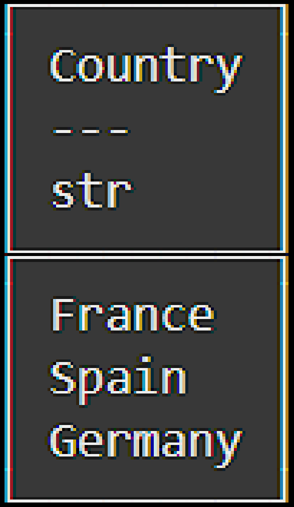
```
Country
---
str

France
Spain
Germany
```

Unique

Returns the unique rows or values from a column, similar to removing duplicates.

```
dataset.select("city").unique()
```



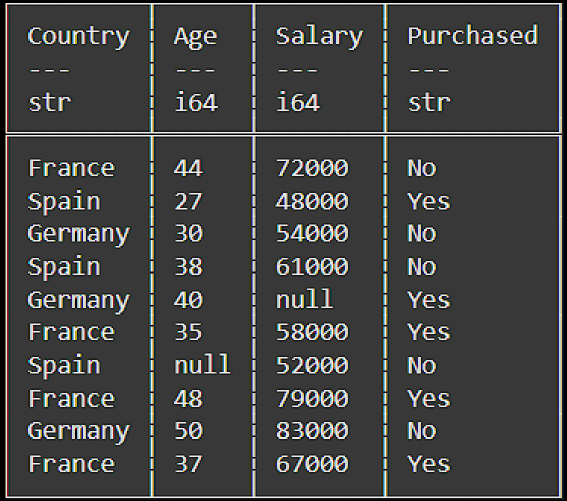


```
Country
---
str

France
Spain
Germany
```

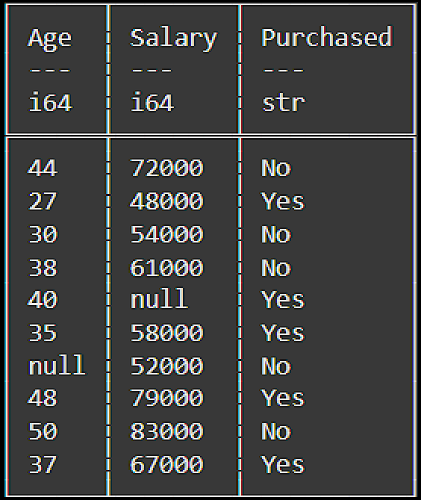
drop

Drops one or more columns from the DataFrame.

```
dataset=dataset.drop("Country")
```



```
Country   Age    Salary   Purchased
---       ---    ---      ---
str       i64    i64      str

France    44     72000    No
Spain     27     48000    Yes
Germany   30     54000    No
Spain     38     61000    No
Germany   40     null     Yes
France    35     58000    Yes
Spain     null   52000    No
France    48     79000    Yes
Germany   50     83000    No
France    37     67000    Yes
```



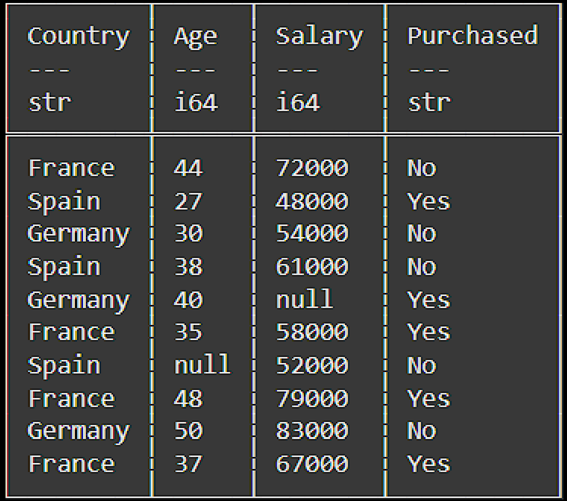
```
Age    Salary   Purchased
---    ---      ---
i64    i64      str

44     72000    No
27     48000    Yes
30     54000    No
38     61000    No
40     null     Yes
35     58000    Yes
null   52000    No
48     79000    Yes
50     83000    No
37     67000    Yes
```

Intelligence Group

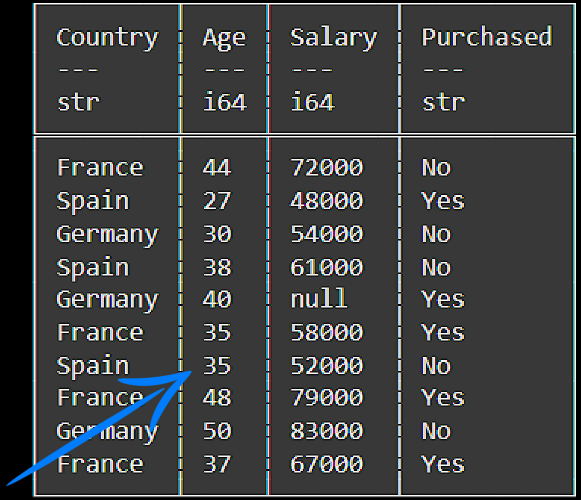
Handling missing values

Use with_columns to update or add columns, select to target a specific column, and fill_null to replace missing values with a desired value.

```
dataset = dataset.with_columns(dataset.select("Age").fill_null(strategy='forward'))
```



```
Country   Age    Salary   Purchased
---       ---    ---      ---
str       i64    i64      str

France    44     72000    No
Spain     27     48000    Yes
Germany   30     54000    No
Spain     38     61000    No
Germany   40     null     Yes
France    35     58000    Yes
Spain     null   52000    No
France    48     79000    Yes
Germany   50     83000    No
France    37     67000    Yes
```



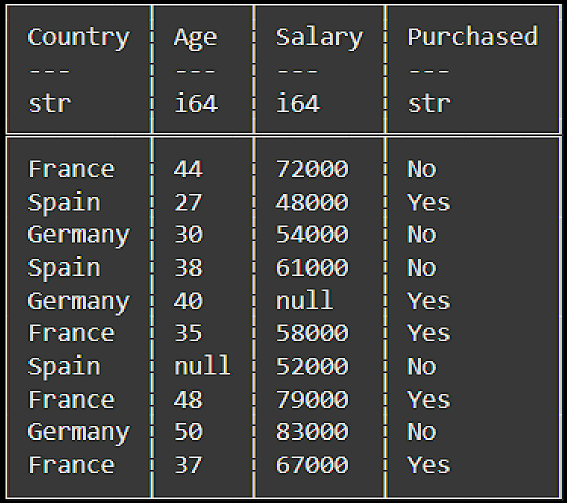
```
Country   Age   Salary   Purchased
---       ---   ---      ---
str       i64   i64      str

France    44    72000    No
Spain     27    48000    Yes
Germany   30    54000    No
Spain     38    61000    No
Germany   40    null     Yes
France    35    58000    Yes
Spain     35    52000    No
France    48    79000    Yes
Germany   50    83000    No
France    37    67000    Yes
```

Intelligence Group

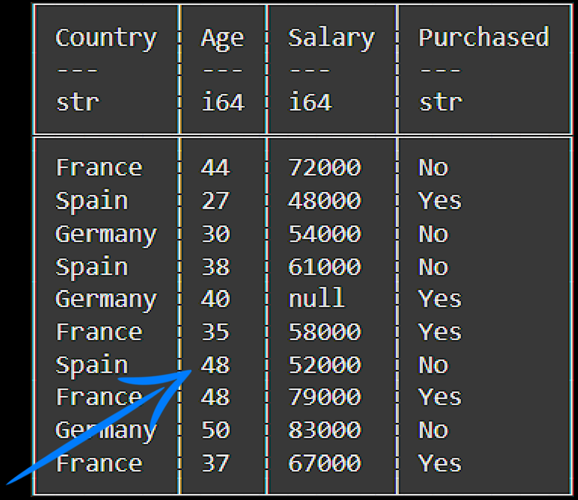
Handling missing values

Use with_columns to update or add columns, select to target a specific column, and fill_null to replace missing values with a desired value.

```
dataset = dataset.with_columns(dataset.select("Age").fill_null(strategy=backward'))
```



```
┌─────────┬──────┬────────┬───────────┐
│ Country ┆ Age  ┆ Salary ┆ Purchased │
│ ---     ┆ ---  ┆ ---    ┆ ---       │
│ str     ┆ i64  ┆ i64    ┆ str       │
╞═════════╪══════╪════════╪═══════════╡
│ France  ┆ 44   ┆ 72000  ┆ No        │
│ Spain   ┆ 27   ┆ 48000  ┆ Yes       │
│ Germany ┆ 30   ┆ 54000  ┆ No        │
│ Spain   ┆ 38   ┆ 61000  ┆ No        │
│ Germany ┆ 40   ┆ null   ┆ Yes       │
│ France  ┆ 35   ┆ 58000  ┆ Yes       │
│ Spain   ┆ null ┆ 52000  ┆ No        │
│ France  ┆ 48   ┆ 79000  ┆ Yes       │
│ Germany ┆ 50   ┆ 83000  ┆ No        │
│ France  ┆ 37   ┆ 67000  ┆ Yes       │
└─────────┴──────┴────────┴───────────┘
```

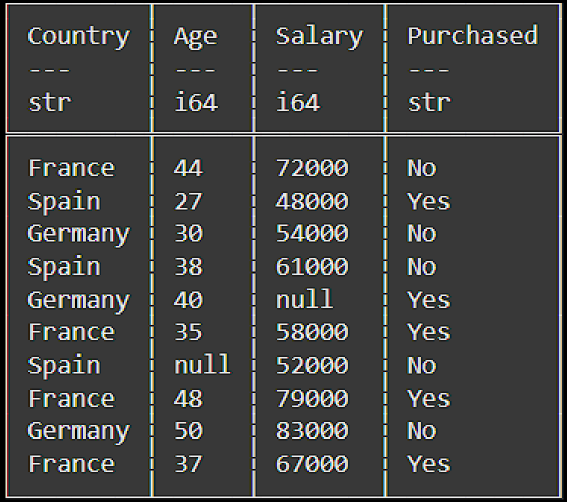


```
┌─────────┬─────┬────────┬───────────┐
│ Country ┆ Age ┆ Salary ┆ Purchased │
│ ---     ┆ --- ┆ ---    ┆ ---       │
│ str     ┆ i64 ┆ i64    ┆ str       │
╞═════════╪═════╪════════╪═══════════╡
│ France  ┆ 44  ┆ 72000  ┆ No        │
│ Spain   ┆ 27  ┆ 48000  ┆ Yes       │
│ Germany ┆ 30  ┆ 54000  ┆ No        │
│ Spain   ┆ 38  ┆ 61000  ┆ No        │
│ Germany ┆ 40  ┆ null   ┆ Yes       │
│ France  ┆ 35  ┆ 58000  ┆ Yes       │
│ Spain   ┆ 48  ┆ 52000  ┆ No        │
│ France  ┆ 48  ┆ 79000  ┆ Yes       │
│ Germany ┆ 50  ┆ 83000  ┆ No        │
│ France  ┆ 37  ┆ 67000  ┆ Yes       │
└─────────┴─────┴────────┴───────────┘
```

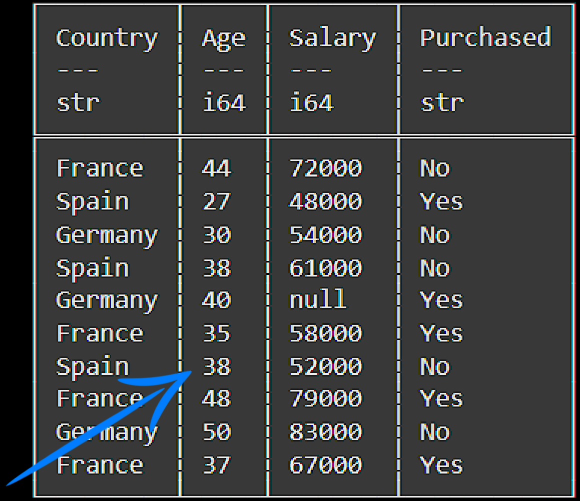
Handling missing values

Use with_columns to update or add columns, select to target a specific column, and fill_null to replace missing values with a desired value.

```
dataset = dataset.with_columns(dataset.select("Age").fill_null(strategy=mean'))
```



```
Country   Age    Salary  Purchased
---       ---    ---     ---
str       i64    i64     str

France    44     72000   No
Spain     27     48000   Yes
Germany   30     54000   No
Spain     38     61000   No
Germany   40     null    Yes
France    35     58000   Yes
Spain     null   52000   No
France    48     79000   Yes
Germany   50     83000   No
France    37     67000   Yes
```

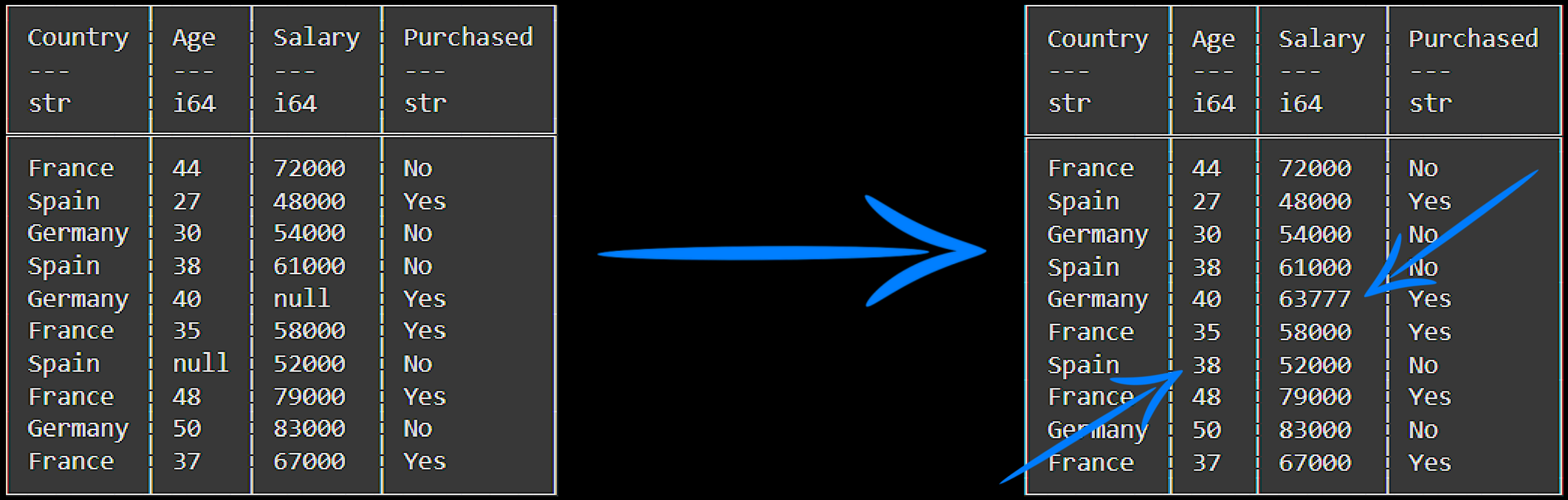


```
Country   Age   Salary  Purchased
---       ---   ---     ---
str       i64   i64     str

France    44    72000   No
Spain     27    48000   Yes
Germany   30    54000   No
Spain     38    61000   No
Germany   40    null    Yes
France    35    58000   Yes
Spain     38    52000   No
France    48    79000   Yes
Germany   50    83000   No
France    37    67000   Yes
```

Handling missing values

Use with_columns to update or add columns, select to target a specific column, and fill_null to replace missing values with a desired value.

```
dataset = dataset.with_columns(dataset.fill_null(strategy=mean'))
```



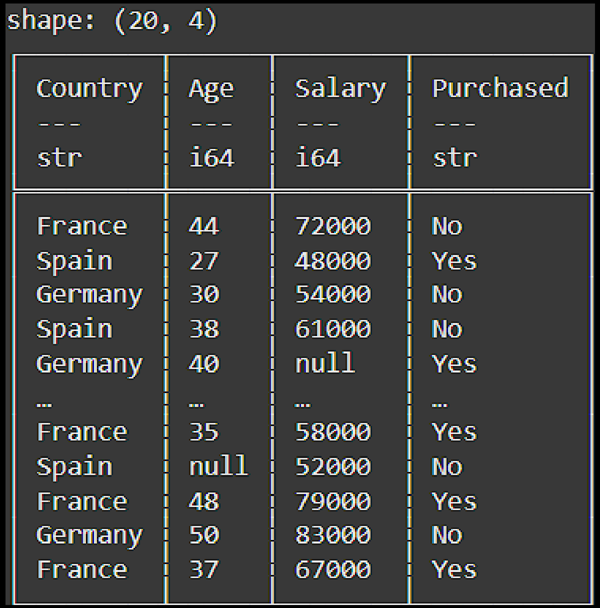
```
Country   Age    Salary   Purchased
---       ---    ---      ---
str       i64    i64      str

France    44     72000    No
Spain     27     48000    Yes
Germany   30     54000    No
Spain     38     61000    No
Germany   40     null     Yes
France    35     58000    Yes
Spain     null   52000    No
France    48     79000    Yes
Germany   50     83000    No
France    37     67000    Yes
```

```
Country   Age   Salary   Purchased
---       ---   ---      ---
str       i64   i64      str

France    44    72000    No
Spain     27    48000    Yes
Germany   30    54000    No
Spain     38    61000    No
Germany   40    63777    Yes
France    35    58000    Yes
Spain     38    52000    No
France    48    79000    Yes
Germany   50    83000    No
France    37    67000    Yes
```

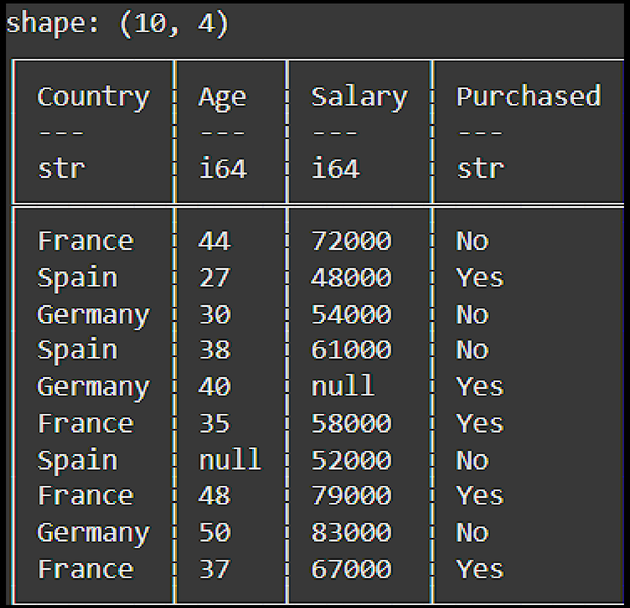
concatenation

Concatenates multiple DataFrames together either vertically (stacking rows) or horizontally (stacking columns).

```
print(dataset)
x = pl.concat([dataset, dataset])
print(x)
```



```
shape: (20, 4)
┌─────────┬──────┬────────┬───────────┐
│ Country ┆ Age  ┆ Salary ┆ Purchased │
│ ---     ┆ ---  ┆ ---    ┆ ---       │
│ str     ┆ i64  ┆ i64    ┆ str       │
╞═════════╪══════╪════════╪═══════════╡
│ France  ┆ 44   ┆ 72000  ┆ No        │
│ Spain   ┆ 27   ┆ 48000  ┆ Yes       │
│ Germany ┆ 30   ┆ 54000  ┆ No        │
│ Spain   ┆ 38   ┆ 61000  ┆ No        │
│ Germany ┆ 40   ┆ null   ┆ Yes       │
│ …       ┆ …    ┆ …      ┆ …         │
│ France  ┆ 35   ┆ 58000  ┆ Yes       │
│ Spain   ┆ null ┆ 52000  ┆ No        │
│ France  ┆ 48   ┆ 79000  ┆ Yes       │
│ Germany ┆ 50   ┆ 83000  ┆ No        │
│ France  ┆ 37   ┆ 67000  ┆ Yes       │
└─────────┴──────┴────────┴───────────┘
```



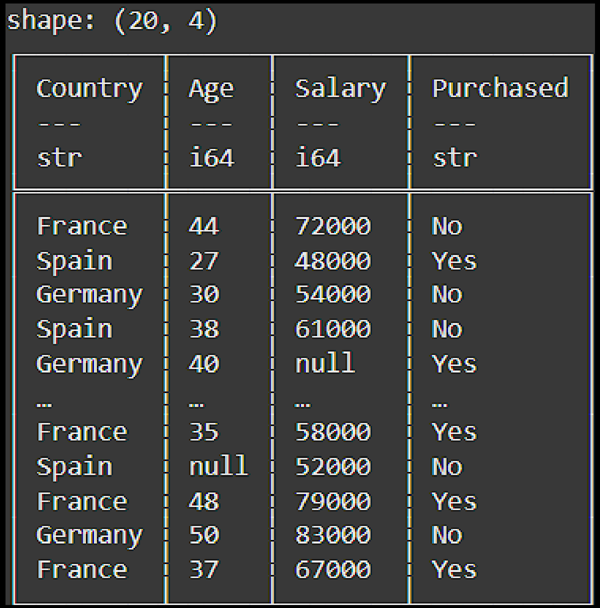
```
shape: (10, 4)
┌─────────┬──────┬────────┬───────────┐
│ Country ┆ Age  ┆ Salary ┆ Purchased │
│ ---     ┆ ---  ┆ ---    ┆ ---       │
│ str     ┆ i64  ┆ i64    ┆ str       │
╞═════════╪══════╪════════╪═══════════╡
│ France  ┆ 44   ┆ 72000  ┆ No        │
│ Spain   ┆ 27   ┆ 48000  ┆ Yes       │
│ Germany ┆ 30   ┆ 54000  ┆ No        │
│ Spain   ┆ 38   ┆ 61000  ┆ No        │
│ Germany ┆ 40   ┆ null   ┆ Yes       │
│ France  ┆ 35   ┆ 58000  ┆ Yes       │
│ Spain   ┆ null ┆ 52000  ┆ No        │
│ France  ┆ 48   ┆ 79000  ┆ Yes       │
│ Germany ┆ 50   ┆ 83000  ┆ No        │
│ France  ┆ 37   ┆ 67000  ┆ Yes       │
└─────────┴──────┴────────┴───────────┘
```

Intelligence Group

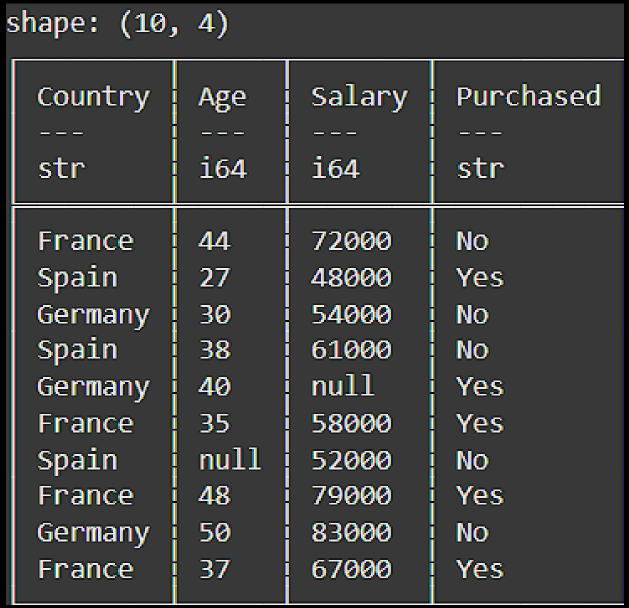
concatenation

To concatenate multiple DataFrames vertically (stacking rows), the datasets must have the same structure, meaning they should contain the same columns or features.

```
print(dataset)
x = pl.concat([dataset, dataset])
print(x)
```



```
shape: (20, 4)
┌─────────┬──────┬────────┬───────────┐
│ Country ┆ Age  ┆ Salary ┆ Purchased │
│ ---     ┆ ---  ┆ ---    ┆ ---       │
│ str     ┆ i64  ┆ i64    ┆ str       │
╞═════════╪══════╪════════╪═══════════╡
│ France  ┆ 44   ┆ 72000  ┆ No        │
│ Spain   ┆ 27   ┆ 48000  ┆ Yes       │
│ Germany ┆ 30   ┆ 54000  ┆ No        │
│ Spain   ┆ 38   ┆ 61000  ┆ No        │
│ Germany ┆ 40   ┆ null   ┆ Yes       │
│ …       ┆ …    ┆ …      ┆ …         │
│ France  ┆ 35   ┆ 58000  ┆ Yes       │
│ Spain   ┆ null ┆ 52000  ┆ No        │
│ France  ┆ 48   ┆ 79000  ┆ Yes       │
│ Germany ┆ 50   ┆ 83000  ┆ No        │
│ France  ┆ 37   ┆ 67000  ┆ Yes       │
└─────────┴──────┴────────┴───────────┘
```



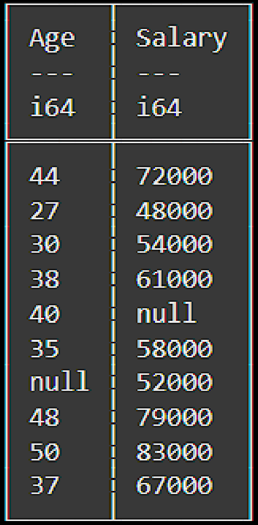
```
shape: (10, 4)
┌─────────┬──────┬────────┬───────────┐
│ Country ┆ Age  ┆ Salary ┆ Purchased │
│ ---     ┆ ---  ┆ ---    ┆ ---       │
│ str     ┆ i64  ┆ i64    ┆ str       │
╞═════════╪══════╪════════╪═══════════╡
│ France  ┆ 44   ┆ 72000  ┆ No        │
│ Spain   ┆ 27   ┆ 48000  ┆ Yes       │
│ Germany ┆ 30   ┆ 54000  ┆ No        │
│ Spain   ┆ 38   ┆ 61000  ┆ No        │
│ Germany ┆ 40   ┆ null   ┆ Yes       │
│ France  ┆ 35   ┆ 58000  ┆ Yes       │
│ Spain   ┆ null ┆ 52000  ┆ No        │
│ France  ┆ 48   ┆ 79000  ┆ Yes       │
│ Germany ┆ 50   ┆ 83000  ┆ No        │
│ France  ┆ 37   ┆ 67000  ┆ Yes       │
└─────────┴──────┴────────┴───────────┘
```

Intelligence Group

concatenation

To concatenate multiple DataFrames horizontally (stacking columns), the datasets should have different columns or features.

```
print(dataset)
x = pl.concat([dataset.select('Age'), dataset.select('Salary')],how='horizontal')
print(x)
```

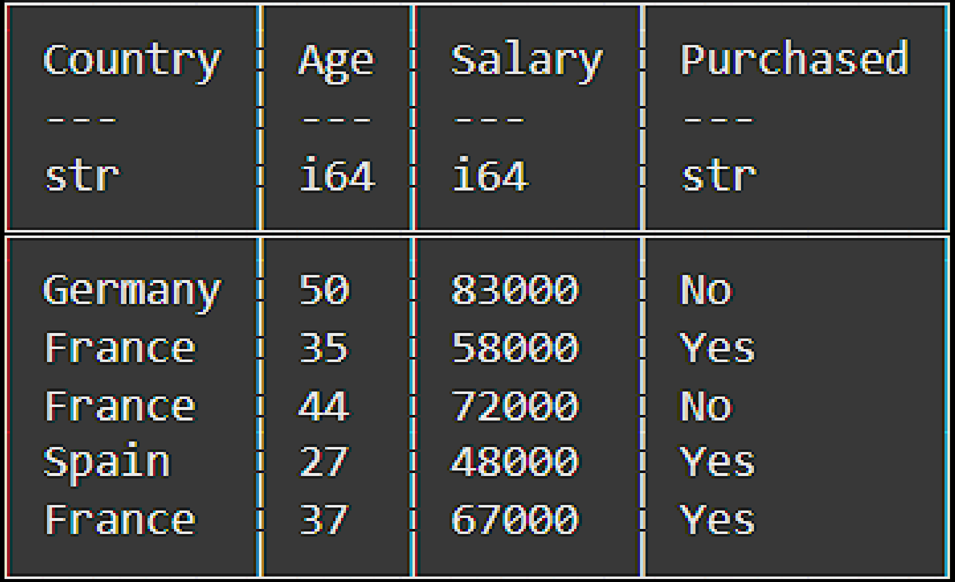


```
Age   Salary
---   ---
i64   i64

44    72000
27    48000
30    54000
38    61000
40    null
35    58000
null  52000
48    79000
50    83000
37    67000
```

sample

Randomly samples rows from the DataFrame, useful for selecting a random subset of the data.

```
print(dataset.sample(n=5))
```



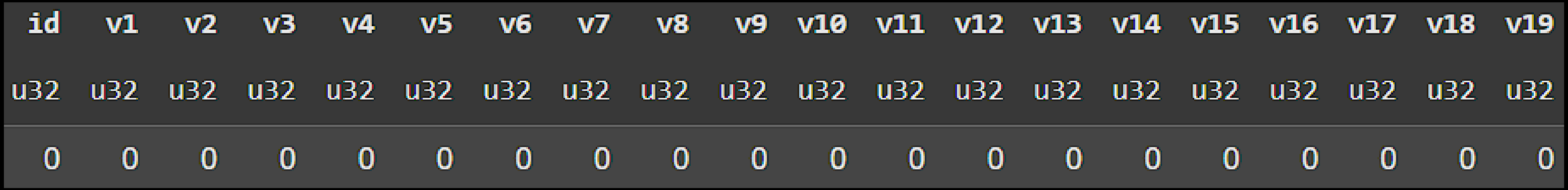
```
┌─────────┬─────┬────────┬───────────┐
│ Country ┆ Age ┆ Salary ┆ Purchased │
│ ---     ┆ --- ┆ ---    ┆ ---       │
│ str     ┆ i64 ┆ i64    ┆ str       │
╞═════════╪═════╪════════╪═══════════╡
│ Germany ┆ 50  ┆ 83000  ┆ No        │
│ France  ┆ 35  ┆ 58000  ┆ Yes       │
│ France  ┆ 44  ┆ 72000  ┆ No        │
│ Spain   ┆ 27  ┆ 48000  ┆ Yes       │
│ France  ┆ 37  ┆ 67000  ┆ Yes       │
└─────────┴─────┴────────┴───────────┘
```

Intelligence Group

Null_counts

The null_count function in Polars is used to calculate the number of null (or missing) values in each column of a DataFrame. It provides a quick and efficient way to check the presence of missing data in your dataset.

```
dataset.null_count()
```



id	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12	v13	v14	v15	v16	v17	v18	v19
u32	u32	u32	u32	u32	u32	u32	u32	u32	u32	u32	u32	u32	u32	u32	u32	u32	u32	u32	u32
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Intelligence Group