# Translating R Packages into Python - A Pilot Project of *EpiDisplay*

## Background:

- Epidemiological analysis is expanding to **new data types** (imaging, genomic, text, etc.)
- **R is widely used, but many epidemiologists prefer Python**
- Our project helps bridge this gap by bringing *EpiDisplay* capabilities into Python.

## **Component Focus**: Importing built-in datasets

### Technologies Reviewed:

| Technology | Primary Role |
|---|---|
| Pandas | General-purpose data analysis and manipulation library |
| Polars | DataFrame library built for speed and scalability |
| Pyreadr | Imports `.rda` / `.rds` files from R into Python |

### Built-in Datasets

- 26 built-in datasets
- 300 Bytes to 173 KB

# Pandas = Python library designed for data manipulation and analysis.

**How it works:**

By importing pandas like this, you can load, clean, and analyze your data efficiently:

```
import pandas as pd

# Load a CSV file into a DataFrame
df = pd.read_csv("data.csv")

# View the first few rows
print(df.head())
```

```
#Import function
import pandas as pd

#Read csv file
pd.read_csv("/Users/marthinmandig/pronto_data/2015_station_data.csv")
```

✓ 0.0s

| | id | name | terminal | lat | long | dockcount | online |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 3rd Ave & Broad St | BT-01 | 47.618418 | -122.350964 | 18 | 10/13/2014 |
| 1 | 2 | 2nd Ave & Vine St | BT-03 | 47.615829 | -122.348564 | 16 | 10/13/2014 |
| 2 | 3 | 6th Ave & Blanchard St | BT-04 | 47.616094 | -122.341102 | 16 | 10/13/2014 |
| 3 | 4 | 2nd Ave & Blanchard St | BT-05 | 47.613110 | -122.344208 | 14 | 10/13/2014 |
| 4 | 5 | 2nd Ave & Pine St | CBD-13 | 47.610185 | -122.339641 | 18 | 10/13/2014 |
| 5 | 6 | 7th Ave & Union St | CBD-03 | 47.610731 | -122.332447 | 20 | 10/13/2014 |
| 6 | 7 | City Hall / 4th Ave & James St | CBD-07 | 47.603509 | -122.330409 | 18 | 10/13/2014 |
| 7 | 8 | Pine St & 9th Ave | SLU-16 | 47.613715 | -122.331777 | 20 | 10/13/2014 |
| 8 | 9 | 2nd Ave & Spring St | CBD-06 | 47.605950 | -122.335768 | 20 | 10/13/2014 |
| 9 | 10 | Summit Ave & E Denny Way | CH-01 | 47.618633 | -122.325249 | 16 | 10/13/2014 |
| 10 | 11 | E Harrison St & Broadway Ave E | CH-02 | 47.622063 | -122.321251 | 20 | 10/13/2014 |
| 11 | 12 | Summit Ave E & E Republican St | CH-03 | 47.623367 | -122.325279 | 16 | 10/13/2014 |
| 12 | 13 | 15th Ave E & E Thomas St | CH-05 | 47.620712 | -122.312805 | 16 | 10/13/2014 |
| 13 | 14 | 12th Ave & E Denny Way | CH-06 | 47.618549 | -122.317017 | 15 | 10/13/2014 |
| 14 | 15 | E Pine St & 16th Ave | CH-07 | 47.615330 | -122.311752 | 18 | 10/13/2014 |
| 15 | 16 | Cal Anderson Park / 11th Ave & Pine St | CH-08 | 47.615486 | -122.318245 | 26 | 10/13/2014 |
| 16 | 17 | Harvard Ave & E Pine St | CH-09 | 47.615517 | -122.322083 | 16 | 10/13/2014 |

**Appeal:**

Simple syntax for loading and managing data, strong data manipulation capabilities, integrates smoothly with other libraries (like NumPy, Matplotlib, and scikit-learn).

**Drawbacks:**

Can be memory-intensive for very large datasets, slower compared to specialized data processing tools, and can require additional optimization for performance.

https://www.w3schools.com/python/pandas/pandas_intro.asp

# Polars: a faster solution

https://docs.pola.rs/user-guide/

What is lazy, and why be lazy?
It could apply automatic query optimization:

- work with larger than memory datasets
- catch schema errors before processing the data

## Basic Features

- Strict with data type
  - more integrity, reduce bugs, data_type output
  - import errors, more arguments
- High Efficiency (e.g. 1,192,827*14 dataset)
  - Polars takes 0.1s
  - Pandas takes 2.2s

```
df_pd = pd.read_csv("NYC_Health.csv")
✓ 2.2s
df_pl = pl.read_csv("NYC_Health.csv")
⊗ 0.5s
df_pl = pl.read_csv("NYC_Health.csv", ignore_errors=True)
✓ 0.1s
```

```
"metadata":
        {"outputType": "error",
        …
        "evalue": "could not parse `\"1,113\"` as dtype `i64` at column 'Discharges'
        …
        setting `ignore_errors` to `True`
        …Original error: ```invalid primitive value found during CSV parsing```" }
```

## Advanced Features

- Eager and **lazy** execution
- Without an explicit index
  - reduce memory, higher efficiency
  - less intuitive

```
df_pl_lazy = pl.LazyFrame(df_pl)
df_pl_lazy.collect().head()
✓ 0.0s
```

shape: (5, 14)

| Year | Facility Id | Facility Name | APR DRG Code | APR Severity of Illness Code | APR DRG Description |
|---|---|---|---|---|---|
| i64 | i64 | str | i64 | i64 | str |
| 2016 | 4 | "Albany Memorial Hospital" | 194 | 1 | "Heart Failure" |
| 2016 | 4 | "Albany Memorial Hospital" | 194 | 2 | "Heart Failure" |

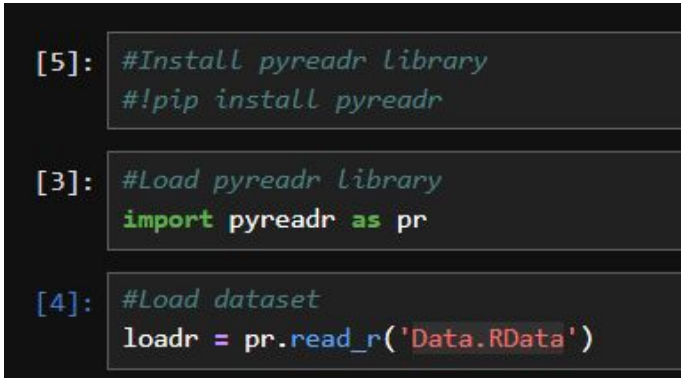
# Pyreadr = Python package that can read Rdata files into or from pandas dataframe**s**

**Appeals**

- Can read R data frames, tibbles, vectors, matrices, arrays, and tables
- Faster than importing via CSV and preserves NA values correctly
- Will return a dictionary with object names as keys and pandas data frames as values

**Limitations**

- Cannot read R lists
- Cannot support data frames with special values like arrays, matrices, and other dataframes
- Can only support writing for a single pandas data frame to a single R data frame
- Pyreadr writing is a slower operation compared to doing it in R




Fajardo O. Pyreadr. Published online December 2018. doi:10.5281/zenodo.7110170

# Summary of 3 Technologies for *Built-in Dataset Importing* Component

*Original build-in datasets are in .RData format, 26 built-in datasets (size 300 Bytes to 173 KB).

| Aspect | pandas | Polars | pyreadr |
|---|---|---|---|
| **Primary Purpose** | General data manipulation and analysis | Fast, parallel DataFrame engine | Interface to load R data formats |
| **Typical Use Case** | Work with CSV, Excel, JSON, SQL | High-speed data wrangling | Extract data stored in R for use in Python |
| **Can Read .rds / .RData**** | No (must convert via R or pyreadr) | No (same as pandas) | **Yes (native support)** |
| **Speed*** (millisec) | Avg 0.58, SD 0.09, Min 0.26, Max 2.01 | Avg 0.55, SD 0.83, Min 0.07, Max 35.41 | Avg 1.88, SD 0.10, Min 0.09, Max 7.14 |
| **Memory Efficiency*** (MB) | Avg 0.30 MB, max 0.57 MB | Avg 0.00 MB, max 0.00 MB | Avg 0.17 MB, max 1.42 MB |
| **Data Type Handling** | Strong (well-tested) | Strong (Arrow-based) | Depends on R conversion (factors, dates) |
| **Conversion Accuracy** | Perfect (for CSVs) | Perfect (for CSVs) | Occasionally requires cleanup |
| **Dependencies** | Light (NumPy core) | Light (Rust backend) | Minimal (C++ bridge to R) |
| **Output Type** | pandas.DataFrame | polars.DataFrame | OrderedDict |
| **Ease of Use** | Very High / Common pandas format | High / Uncommon format | Medium / need dict to DataFrame convert |