POLARS from the beginning

Jonny Edwards @mlabs

https://tinyurl.com/pydatapolars

Plan #1 (total time 3 * 30mins)

- Getting Started (5 mins)
- Reading files 2 ways (5 mins)
    - Dealing with Nulls
- The standard “composable” format (5 mins)
    - examples of:
        - *filter*
        - *select*
        - *with_columns* (10 mins)
    - ***Break (5mins)***

Plan #2

- Data types (Rust) (5 mins)
- Intro to the trainers data-set (Citizen me)
- *ASIDE #1*: **What do I want out of my data** (5 mins)
- *ASIDE #2*: **Bokeh** graph plotting (5 mins)
- Some useful functions (10 mins)
    - apply etc
- ***Break (5 mins)***

Plan #3

- Freeform analysis (20 mins)
- Eager and Lazy
    - if things get too big (5 mins)
- Any questions? (5 mins)

TOTAL 90 minutes

What I want to happen…

1. I have a rough notebook with some examples
2. I want you to be in the data science starting blocks!
3. We are **not** doing this the traditional way of wading through my notebook 😄
    1. Get the data
    2. Open a **new** notebook
    3. Start your own exploration by copy/paste

POLARS == better pandas

- Are we all familiar with the idea of a dataframe?
  - Column oriented data structure
    - Useful more general data work
    - All started with R dataframe
      - Back in the day … (I am that old!)
  - In the case of POLARS rust based (ffi)
    - There is also a javascript version

POLARS is better because

- Fast https://www.pola.rs/benchmarks.html
- Neater about data
- Better API
- Does less to do more

Getting Started CSV Import

first things first POLARS is a little stricter

```
[ ] #imports for data processing
    import polars as pl
    import numpy as np
```

```
#reading csv files
data_path = "drive/My Drive/Data/"

df = pl.read_csv(data_path+"rawtrainers2.csv")

#DAMN...errors!!
```

```
-------------------------------------------------
ComputeError                              Traceba
<ipython-input-5-c9ef40eef5a4> in <cell line: 4>(
      2 data_path = "drive/My Drive/Data/"
      3
----> 4 df = pl.read_csv(data_path+"rawtrainers2.
      5
      6 #DAMN...errors!!
```

```
#try again
df = pl.read_csv(data_path+"rawtrainers2.csv", null_values= ["Enter an answer","-99"])
```

```
df.head()
```

Let’s set the null values

```
#lazy file reading
df = pl.scan_csv(data_path+"rawtrainers2.csv", null_values= ["Enter an answer","-99"])
```

Reading files another way (lazy)

- the workflow is really nice as scan and read work the same
- Schemas can be applied

The standard “composable” pattern

- Three major lifting tools (T)
  - *select* - ***new columns***
  - *with_columns* -**add columns**
  - *filter* - ***new_rows***
- THE FORMAT df.<T>([pl.col**(“foo”).something])**
  - Much easier than pandas! (YMMV)
  - Get used to doing this (there is a Series way)
  - the trick is the **something** *runs in parallel*

The standard “composable” pattern

- **This is the one thing to learn today**
- Gone are the pandas guesswork and sequential processing

3 Examples

```
df = df.with_columns([pl.col('50 Apps Installed on Smartphone-Banking').cast(pl.Boolean).alia
          pl.col('50 Apps Installed on Smartphone-Fitness or sports').cast(pl.Boolean).alias(
          pl.col('50 Apps Installed on Smartphone-Food and drink').cast(pl.Boolean).alias("fo
          pl.col('50 Apps Installed on Smartphone-Grocery shopping').cast(pl.Boolean).alias('
          pl.col('50 Apps Installed on Smartphone-Retail shopping').cast(pl.Boolean).alias("r
          pl.col('50 Apps Installed on Smartphone-Social network').cast(pl.Boolean).alias("so
           ])
```

```
df_running = df.select([pl.col('Average Daily Step Count').cast(pl.UInt16).alias("daystep
                        pl.col('Average Daily Step Count RC').cast(pl.UInt16).alias("days
                        pl.col('Average Weekly Step Count').cast(pl.UInt32).alias("weekste
                        pl.col('Running Frequency').cast(pl.UInt32).alias("freq")])
df_running.describe()
```

```
[ ]  df_running.filter(pl.col("weekstep")!=0).descri
```

Datatypes (Rust)

- Polars is actually a Rust library
- Managing datatypes is good practice

Numeric

`Decimal`	Decimal 128-bit type with an optional precision and non-negative scale.
`Float32`	32-bit floating point type.
`Float64`	64-bit floating point type.
`Int8`	8-bit signed integer type.
`Int16`	16-bit signed integer type.
`Int32`	32-bit signed integer type.
`Int64`	64-bit signed integer type.
`UInt8`	8-bit unsigned integer type.
`UInt16`	16-bit unsigned integer type.
`UInt32`	32-bit unsigned integer type.
`UInt64`	64-bit unsigned integer type.

Datatypes (Rust)

Temporal

`Date`	Calendar date type.
`Datetime`	Calendar date and time type.
`Duration`	Time duration/delta type.
`Time`	Time of day type.

Nested

`Array`(*args, **kwargs)
`List`(*args, **kwargs)
`Struct`(*args, **kwargs)

Intro to the trainers data-set (Citizen me)

Delivered as a part of the CitizenMe labs project

```
[ ]  df.columns

    ['RespondentID',
     'CollectorID',
     'Completion Date',
     'ExternalId',
     'Country of Residence',
     'Average Daily Step Count RC',
     'Average Daily Step Count',
     'Average Weekly Step Count RC',
     'Average Weekly Step Count',
     'Running Frequency',
     '50 Apps Installed on Smartphone-Banking',
     '50 Apps Installed on Smartphone-Revoult bank',
```

ASIDE #1: What do I want out of my data?

- I use Polars as my goto for data analysis
  - Some exploratory stuff
  - Data led insight
  - Some forms of hypothesis driven
  - **THEN** present findings
- The story for an ETL pipeline is even stronger

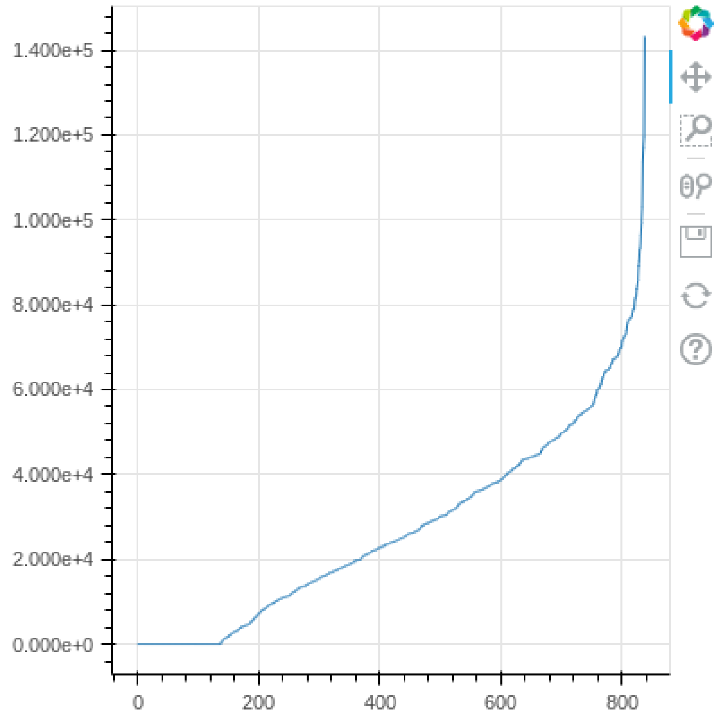
ASIDE #2: Bokeh graph plotting

- This eventually means we need to do some visualisation
- Interesting talk at ODSC
    - Information led
    - knowledge led
- I like **Bokeh** - but not one of the commonly used graphing libraries

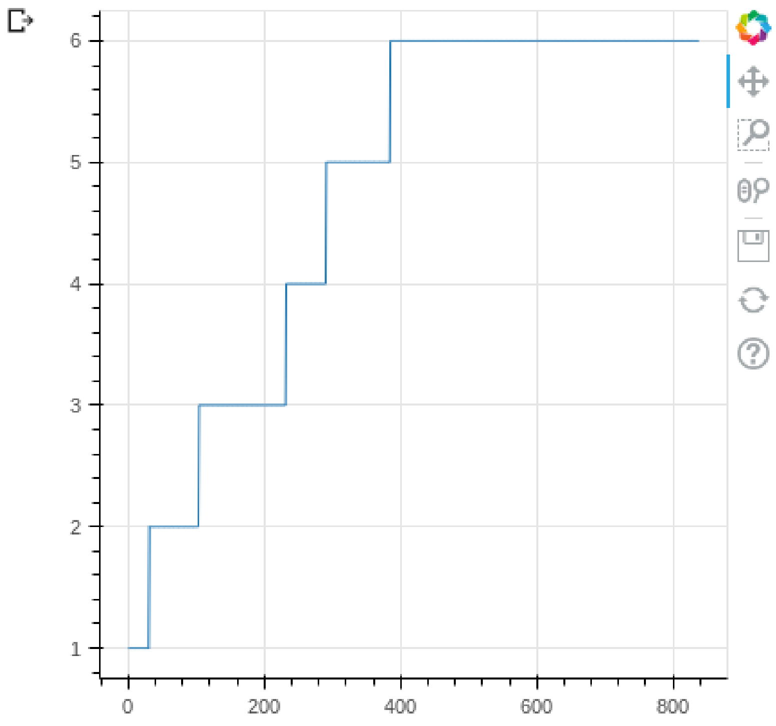
```
from bokeh.io import output_notebook, show
output_notebook()
from bokeh.plotting import figure
p = figure(width=400, height=400)
p.line(range(len(df_running["weekstep"])),df_running["weekstep"])
show(p) # show the results
```

A typical output

- Clean
- Some people don't do steps
- A small number do a lot
- *Is that insight?*



same but perceived frequency



Some useful functions

Functionality very similar to Pandas

- A bit of pandas knowledge is useful
- The documentation is really really good

```
df_running.filter(pl.col("weekstep").is_not_null()).corr()
```

shape: (2, 2)

weekstep	freq
f64	f64
1.0	-0.039903
-0.039903	1.0

groupby is really useful and follows the same composable pattern

- Males running do more steps than females

```
df_running.groupby("gender").agg([pl.col("freq").mean(),pl.col("weekstep").mean()])
```

shape: (2, 3)

gender	freq	weekstep
u8	f64	f64
1	4.361111	29814.024691
2	5.005825	25671.048638

apply in a with_columns

```python
def level(x):
  if 33 < x < 66:
    return 2
  elif x < 33:
    return 1
  elif x > 66:
    return 3

df_running = df_running.with_columns([pl.col("Thrill-Seeking").apply(level).alias("thrill"),
                                      pl.col('Conscientious').apply(level).alias("con"),
                                      pl.col('Extraverted').apply(level).alias("extra")])

df_running.groupby("thrill").agg([pl.col("weekstep").mean()])
```

shape: (4, 2)

thrill	weekstep
i64	f64
2	28709.18
1	22710.175966
3	29429.816327
null	39734.75

Looking at trainer popularity

```python
train_cols = [(col,col.split("?")[1].strip()) for col in df.columns if col[:2]=="Q7"]
df_trainers = df.select([pl.col(t_in).alias(t_out).cast(pl.Boolean) for t_in,t_out in train_cols])
df_trainers.select([pl.col("*").value_counts()])
```

shape: (2, 25)

Nike	adidas	Converse	Vans	Puma	Reebok	New Balance	Fila	Asics	Under Armour	
struct[2]	struct[2]	struct[2]	struct[2]	struct[2]	struct[2]	struct[2]	struct[2]	struct[2]	struct[2]	str
{true,551}	{true,485}	{true,283}	{false,595}	{true,163}	{true,117}	{false,708}	{true,62}	{false,770}	{false,766}	{fal
{false,288}	{false,354}	{false,556}	{true,244}	{false,676}	{false,722}	{true,131}	{false,777}	{true,69}	{true,73}	{t

Looking at trainer popularity by other metrics (semi successful)

- Series
- iter_rows
- explode
- hstack
- value_counts

```
trainers = x = pl.Series("trainer", [[df_trainers.columns[i] for i,j in enumerate(k) if j] for k in df_trainers.iter_rows()])
df_trainers_shoe = df_running.hstack([trainers]).explode("trainer")
df_sum = df_trainers_shoe.groupby("trainer").agg([pl.col("freq").value_counts(),pl.col("weekstep").mean()])
df_sum
```

shape: (25, 3)

trainer	freq	weekstep
str	list[struct[2]]	f64
"Lotto"	[{6,3}, {2,1}, {5,2}]	40598.666667
"Puma"	[{2,17}, {3,20}, ... {6,97}]	28029.93865
"New Balance"	[{3,21}, {2,15}, ... {4,3}]	29028.053435
"Under Armour"	[{1,3}, {5,10}, ... {4,1}]	32349.902778
"adidas"	[{2,40}, {5,49}, ... {4,33}]	27906.452479

Your Time (20 mins)

Lazy and Eager

- filter can happen before reading the data
- means memory is managed
- particularly useful for large data-set
  - My success story!
    - geosearching for MI applications

Finally….

- Yes?
- No way back now 👍
    - Are you joining the Polars Revolutionary Army?
- Confession…