

# Scale your workflow beyond memory limits with Arrow

Dr Mike Spencer



# Are your data getting bigger?



# Is your computer?



# The problem

- More data
- Larger data
- Extra data



# The problem

- More data
- Larger data
- Extra data

...same resources





```
> read_csv("data_in/finance_2019-01-06.csv") %>% glimpse()
```

```
Rows: 500000 Columns: 19
```

```
— Column specification —
```

```
Delimiter: ","
```

```
chr   (2): cid, sex
```

```
dbl  (16): income, income_salary, income_benefits, income_pension, income_investment, income_interest, income_other, expend...
```

```
date  (1): end_of_this_period
```

```
i Use `spec()` to retrieve the full column specification for this data.
```

```
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
Rows: 500,000
```

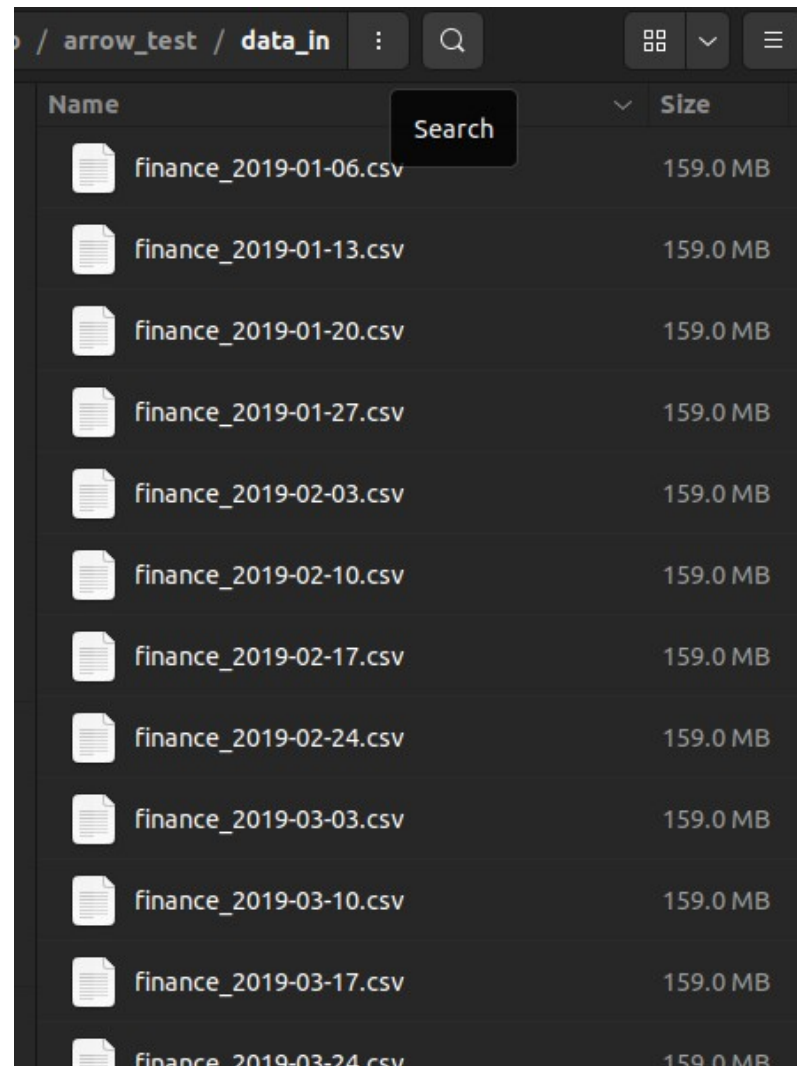
```
Columns: 19
```

```
$ cid           <chr> "0372943541", "8718939101", "9576403582", "6434287673", "8384245418", "1877660945", "872680...
$ sex           <chr> "M", "F", "F", "F", "F", "M", "M", "F", "F", "F", "F", "M", "M", "F", "M", "M", "M", "F", "...
$ end_of_this_period <date> 2019-01-06, 2019-01-06, 2019-01-06, 2019-01-06, 2019-01-06, 2019-01-06, 2019-01-06, 2019-0...
$ income        <dbl> 62384.298, 3984.488, 13459.366, 102496.911, 14291.991, 16236.704, 25905.508, 10583.545, 140...
$ income_salary <dbl> 12821.127, 59310.121, 17909.762, 13447.458, 16197.562, 26368.778, 18953.949, 78661.545, 299...
$ income_benefits <dbl> 1030.0490, 9868.1515, 1187.4295, 4605.2130, 743.6535, 834.0886, 2441.3281, 1011.7422, 2120...
$ income_pension <dbl> 12814.960, 38493.055, 8582.256, 24415.374, 6362.781, 1003.522, 5742.224, 7761.511, 3404.030...
$ income_investment <dbl> 69.43343, 90.77197, 263.00960, 78.18585, 26.86534, 59.99697, 198.48226, 52.14309, 161.82220...
$ income_interest <dbl> 31.69776, 385.90542, 23.60480, 188.74733, 367.46442, 103.29140, 78.88642, 367.41628, 98.814...
$ income_other    <dbl> 1355.6848, 28890.9560, 2577.5952, 28988.3237, 5731.3177, 872.0811, 19111.8576, 1122.4333, 3...
$ expenditure     <dbl> 59265.083, 3785.264, 12786.398, 97372.065, 13577.391, 15424.869, 24610.233, 10054.367, 1335...
$ expenditure_committed <dbl> 18715.289, 1195.346, 4037.810, 30749.073, 4287.597, 4871.011, 7771.652, 3175.063, 4218.116,...
$ expenditure_essential <dbl> 31192.149, 1992.244, 6729.683, 51248.455, 7145.995, 8118.352, 12952.754, 5291.772, 7030.194...
$ expenditure_qol  <dbl> 6238.4298, 398.4488, 1345.9366, 10249.6911, 1429.1991, 1623.6704, 2590.5508, 1058.3545, 140...
$ expenditure_discretionary <dbl> 3119.2149, 199.2244, 672.9683, 5124.8455, 714.5995, 811.8352, 1295.2754, 529.1772, 703.0194...
$ expenditure_uncategorized <dbl> 3119.2149, 199.2244, 672.9683, 5124.8455, 714.5995, 811.8352, 1295.2754, 529.1772, 703.0194...
$ cash_balance_final <dbl> 308.4810, 332.9998, 332.8288, 317.8296, 237.9787, 314.0453, 263.2046, 346.5375, 324.4015, 4...
$ cash_min        <dbl> 99.71669, 142.70067, 139.40952, 93.90980, 155.48634, 99.35683, 133.75753, 118.08549, 79.321...
$ cash_max        <dbl> 965.3130, 1056.2350, 1029.5984, 980.3929, 941.5497, 1011.3531, 948.6580, 1068.6138, 994.693...
```

# Toy data

# Toy data

- 228 weeks of data
- 500,000 'people'
- 19 columns
- 36 GB total



Name	Size
finance_2019-01-06.csv	159.0 MB
finance_2019-01-13.csv	159.0 MB
finance_2019-01-20.csv	159.0 MB
finance_2019-01-27.csv	159.0 MB
finance_2019-02-03.csv	159.0 MB
finance_2019-02-10.csv	159.0 MB
finance_2019-02-17.csv	159.0 MB
finance_2019-02-24.csv	159.0 MB
finance_2019-03-03.csv	159.0 MB
finance_2019-03-10.csv	159.0 MB
finance_2019-03-17.csv	159.0 MB
finance_2019-03-24.csv	159.0 MB



# Challenge

For each week:

- Mean income
- Count accounts with a negative balance
- Split by sex



# What would you do?



# Partition/chunk data

- 10 equal groups, based on 1<sup>st</sup> cid digit
- Process intensive to crawl through data by week and split into groups
- Could then calculate challenge using idgroup files (10% of total size)



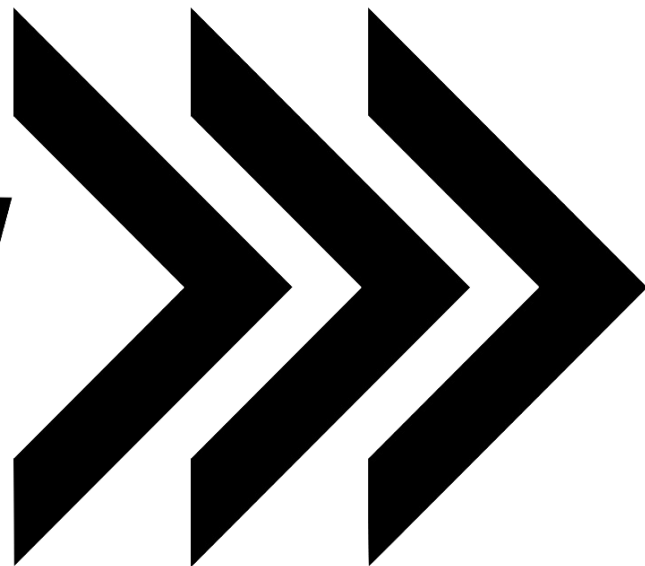
# What about median?



# Introducing

APACHE

# ARROW



# <https://arrow.apache.org/>

Apache Arrow defines a language-independent columnar memory format for flat and hierarchical data, organized for efficient analytic operations on modern hardware like CPUs and GPUs. The Arrow memory format also supports zero-copy reads for lightning-fast data access without serialization overhead.



# <https://arrow.apache.org/>

Apache Arrow defines a language-independent columnar memory format for flat and hierarchical data, organized for **efficient** analytic operations on modern hardware like CPUs and GPUs. The Arrow memory format also supports zero-copy reads for **lightning-fast** data access without serialization overhead.



# Resources



Device Name	SB-ThinkPad-L390 >
Hardware Model	Lenovo ThinkPad L390
Memory	16.0 GiB
Processor	Intel® Core™ i5-8265U CPU @ 1.60GHz × 8
Graphics	Mesa Intel® UHD Graphics 620 (WHL GT2)
Disk Capacity	256.1 GB



# File/table vs dataset

## Dataset vs file

- `read_csv` similar to `read_csv_arrow`
- `open_dataset` similar to a db connection

## csv vs parquet

- csv – comma separated value
- parquet – column orientated format with metadata, from Hadoop ecosystem





# Limitations

- Some functions not implemented (e.g. cut)
- Need to have partitions suited to task
  - Time
  - Location
  - Other



# Summary

- Arrow is very fast
- Can work on data bigger than memory
- Good partitions are key
- Use with `dplyr`
- [github.com/mikerspencer/arrow\\_test](https://github.com/mikerspencer/arrow_test)



# Contact

- [michael.spencer@smartdatafoundry.com](mailto:michael.spencer@smartdatafoundry.com)
- [@mikerspencer@mastodon.scot](https://masto.host/mikerspencer)
- [linkedin.com/in/mikerspencer/](https://www.linkedin.com/in/mikerspencer/)
- [@mikerspencer:matrix.org](https://matrix.to/#/@mikerspencer:matrix.org)
- [smartdatafoundry.com](https://smartdatafoundry.com)