PyConChina 2019

人人都爱 DataFrame

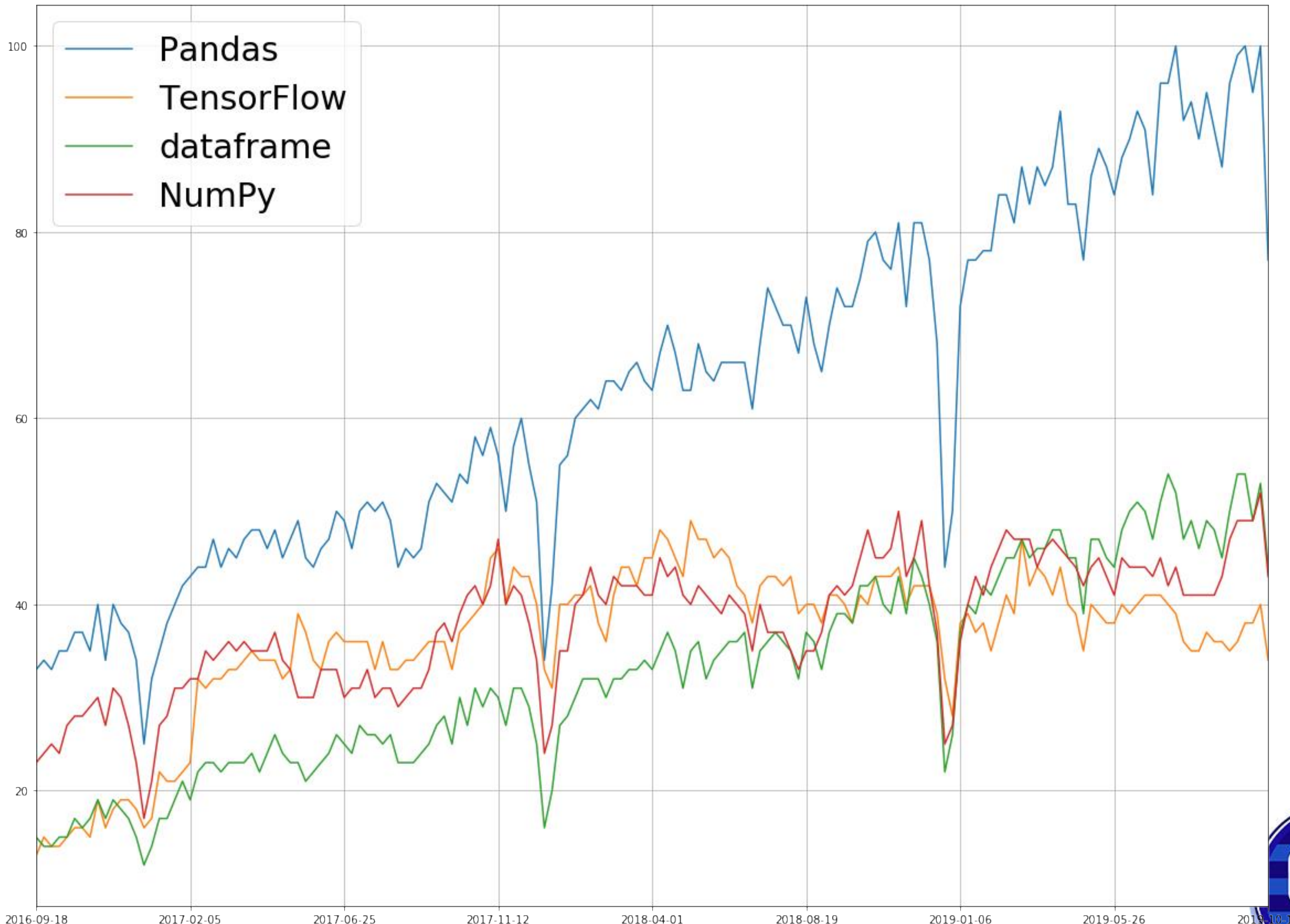
Pandas 到 Mars 的进阶之路

何开圣

目录 CONTENTS

1 Python 数据科学生态

Google Trends



Python 数据科学栈

scikit learn

TensorFlow

PyTorch

mxnet

SciPy

pandas

NumPy

PYTHON 30th

伴随 Pandas 的标签

Pythonic

完善的 API

数据分析

DataFrame

开源

强大的社区

数据清洗

什么是 Pandas

PyConChina 2019

Series

- 一维数组数据
- 一个与数组关联的数据标签（索引）

```
In [12]: s = pd.Series([1, 3, 5, np.nan, 6, 8], name='col')

In [13]: s
Out[13]:
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
Name: col, dtype: float64
```

DataFrame

- 表格数据
- 二维有标签的数据
- 看作是一个Series的字典（所有Series共享一个索引）

```
In [8]: df
Out[8]:
                   A         B         C         D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401
2013-01-06 -0.673690  0.113648 -1.478427  0.524988
```

Pandas 可以做什么

PyConChina 2019

IO tools

- CSV & text files
- JSON
- HTML
- Excel files
- OpenDocument Spreadsheets
- ……

Indexing and selecting data

- Attribute access
- Slicing ranges
- Selection by label
- Selection by position
- Selection by callable
- ……

Group By: split-apply-combine

- Splitting an object into groups
- Iterating through groups
- Selecting a group
- Aggregation
- Transformation
- ……

Merge, join, and concatenate

- Concatenating objects
- Database-style DataFrame or named Series joining/merging
- Timeseries friendly merging

Computational tools

- Statistical functions
- Window Functions
- Aggregation
- Expanding windows
- Exponentially weighted windows

- **Visualization**
- **Working with text data**
- **Working with missing data**
- **Sparse data structure**

PYTHON 30th

走进 Pandas

创建对象

```
In [1]: import pandas as pd

In [2]: import numpy as np

In [3]: dates = pd.date_range('20191019', periods=6)

In [4]: df = pd.DataFrame(np.random.rand(6, 4), index=dates,
columns=list('abcd'))

In [5]: df
Out[5]:
                   a         b         c         d
2019-10-19  0.343870  0.597608  0.873940  0.419012
2019-10-20  0.760962  0.296669  0.315243  0.836361
2019-10-21  0.294613  0.652080  0.534011  0.484261
2019-10-22  0.728239  0.928620  0.595617  0.557358
2019-10-23  0.031837  0.029126  0.982403  0.459137
2019-10-24  0.290760  0.936916  0.416385  0.646032
```

走进 Pandas

查看数据

```
In [8]: df.head(3)
Out[8]:
                   a         b         c         d
2019-10-19  0.343870  0.597608  0.873940  0.419012
2019-10-20  0.760962  0.296669  0.315243  0.836361
2019-10-21  0.294613  0.652080  0.534011  0.484261

In [9]: df.tail(3)
Out[9]:
                   a         b         c         d
2019-10-22  0.728239  0.928620  0.595617  0.557358
2019-10-23  0.031837  0.029126  0.982403  0.459137
2019-10-24  0.290760  0.936916  0.416385  0.646032
```

```
In [10]: df.index
Out[10]:
DatetimeIndex(['2019-10-19',
'2019-10-20', '2019-10-21',
'2019-10-22',
               '2019-10-23',
'2019-10-24'],

dtype='datetime64[ns]', freq='D')

In [11]: df.columns
Out[11]: Index(['a', 'b', 'c',
'd'], dtype='object')

In [12]: df.dtypes
Out[12]:
a     float64
b     float64
c     float64
d     float64
dtype: object
```

走进 Pandas

PyConChina 2019

查看数据

```
In [13]: df.describe()
Out[13]:
              a         b         c         d
count  6.000000  6.000000  6.000000  6.000000
mean   0.408380  0.573503  0.619600  0.567027
std    0.282612  0.357188  0.260056  0.154451
min    0.031837  0.029126  0.315243  0.419012
25%    0.291723  0.371904  0.445792  0.465418
50%    0.319242  0.624844  0.564814  0.520810
75%    0.632147  0.859485  0.804359  0.623863
max    0.760962  0.936916  0.982403  0.836361
```

```
In [14]: df.sort_values(by='c')
Out[14]:
                   a         b         c         d
2019-10-20  0.760962  0.296669  0.315243  0.836361
2019-10-24  0.290760  0.936916  0.416385  0.646032
2019-10-21  0.294613  0.652080  0.534011  0.484261
2019-10-22  0.728239  0.928620  0.595617  0.557358
2019-10-19  0.343870  0.597608  0.873940  0.419012
2019-10-23  0.031837  0.029126  0.982403  0.459137
```

PYTHON 30th

走进 Pandas

选择数据

```
In [15]: df['a']
Out[15]:
2019-10-19    0.343870
2019-10-20    0.760962
2019-10-21    0.294613
2019-10-22    0.728239
2019-10-23    0.031837
2019-10-24    0.290760
Freq: D, Name: a, dtype: float64

In [16]: df[:2]
Out[16]:
                   a         b         c         d
2019-10-19  0.343870  0.597608  0.873940  0.419012
2019-10-20  0.760962  0.296669  0.315243  0.836361
```

```
In [17]: df.loc['2019-10-20':'2019-10-22', ['b', 'c']]
Out[17]:
                   b         c
2019-10-20  0.296669  0.315243
2019-10-21  0.652080  0.534011
2019-10-22  0.928620  0.595617

In [18]: df.iloc[3:5,0:2]
Out[18]:
                   a         b
2019-10-22  0.728239  0.928620
2019-10-23  0.031837  0.029126
```

走进 Pandas

PyConChina 2019

选择数据

```
In [19]: df[df['a'] > 0.5]
Out[19]:
                   a         b         c         d
2019-10-20  0.760962  0.296669  0.315243  0.836361
2019-10-22  0.728239  0.928620  0.595617  0.557358

In [20]: df[df < 0.1]
Out[20]:
                   a         b   c   d
2019-10-19       NaN       NaN NaN NaN
2019-10-20       NaN       NaN NaN NaN
2019-10-21       NaN       NaN NaN NaN
2019-10-22       NaN       NaN NaN NaN
2019-10-23  0.031837  0.029126 NaN NaN
2019-10-24       NaN       NaN NaN NaN
```

PYTHON 30th

走进 Pandas

PyConChina 2019

一些计算

```
In [21]: df.mean()
Out[21]:
a     0.408380
b     0.573503
c     0.619600
d     0.567027
dtype: float64

In [22]: df.mean(axis=1)
Out[22]:
2019-10-19    0.558608
2019-10-20    0.552309
2019-10-21    0.491241
2019-10-22    0.702459
2019-10-23    0.375626
2019-10-24    0.572523
Freq: D, dtype: float64
```

```
In [29]: s = pd.Series([1, 2, 3],
index=df.index[2:5])

In [30]: s
Out[30]:
2019-10-21    1
2019-10-22    2
2019-10-23    3
Freq: D, dtype: int64

In [31]: df.add(s, axis='index')
Out[31]:
                   a         b         c         d
2019-10-19       NaN       NaN       NaN       NaN
2019-10-20       NaN       NaN       NaN       NaN
2019-10-21  1.294613  1.652080  1.534011  1.484261
2019-10-22  2.728239  2.928620  2.595617  2.557358
2019-10-23  3.031837  3.029126  3.982403  3.459137
2019-10-24       NaN       NaN       NaN       NaN
```

PYTHON 30th

走进 Pandas

合并 (SQL join)

```
In [32]: left = pd.DataFrame({'key':
['foo', 'foo'], 'lval': [1, 2]})

In [33]: right = pd.DataFrame({'key':
['foo', 'foo'], 'rval': [4, 5]})

In [34]: left
Out[34]:
   key  lval
0  foo     1
1  foo     2

In [35]: right
Out[35]:
   key  rval
0  foo     4
1  foo     5

In [36]: pd.merge(left, right, on='key')
Out[36]:
   key  lval  rval
0  foo     1     4
1  foo     1     5
2  foo     2     4
3  foo     2     5
```

```
In [37]: left = pd.DataFrame({'key':
['foo', 'bar'], 'lval': [1, 2]})

In [38]: right = pd.DataFrame({'key':
['foo', 'bar'], 'rval': [4, 5]})

In [39]: left
Out[39]:
   key  lval
0  foo     1
1  bar     2

In [40]: right
Out[40]:
   key  rval
0  foo     4
1  bar     5

In [41]: pd.merge(left, right, on='key')
Out[41]:
   key  lval  rval
0  foo     1     4
1  bar     2     5
```

走进 Pandas

PyConChina 2019

分组 (groupby)

```
In [42]: df
Out[42]:
     A      B         C         D
0  foo    one -0.449026  0.642683
1  bar    one -0.074552 -0.410214
2  foo    two -0.102928 -1.136713
3  bar  three  0.102498  0.836924
4  foo    two -0.543563 -0.322754
5  bar    two  0.648193 -0.510235
6  foo    one  0.311538 -0.064786
7  foo  three -1.135822 -1.399072

In [43]: df.groupby('A').sum()
Out[43]:
            C          D
A
bar   0.676139   -0.083525
foo  -1.919801   -2.689642
```

- 分组
- 每个分组应用某个 function
- 融合最后的结果

PYTHON 30th

Pandas 周边生态 —— Pandas profiling

describe() 实现了基础的统计信息， pandas-profiling 是数据分析的神器，满足了大部分的统计需求

不再需要纠结使用什么函数，分析哪些数据，直接在网站展示

```
import pandas as pd
import pandas_profiling
data = pd.read_csv("")
profile = pandas_profiling.ProfileReport(data)
profile.to_file('report.html')
```

Pandas profiling

Overview：数据量，数据类型，warning（缺失值以及零值）

Dataset info

Number of variables	14
Number of observations	45726
Missing cells	29703 (4.6%)
Duplicate rows	0 (0.0%)
Total size in memory	4.6 MiB
Average record size in memory	105.0 B

Variables types

Numeric	4
Categorical	5
Boolean	1
Date	1
URL	0
Text (Unique)	1
Rejected	2
Unsupported	0

Warnings

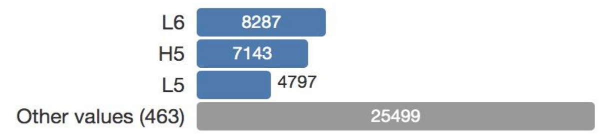
`GeoLocation` has a high cardinality: 17101 distinct values	Warning
`GeoLocation` has 7315 (16.0%) missing values	Missing
`mass_(g)` is highly skewed (γ1 = 76.91847245)	Skewed
`recclass` has a high cardinality: 466 distinct values	Warning
`reclat` has 6438 (14.1%) zeros	Zeros
`reclat` has 7315 (16.0%) missing values	Missing
`reclat_city` is highly correlated with `reclat` (ρ = 0.9942518712)	Rejected
`reclong` has 6214 (13.6%) zeros	Zeros
`reclong` has 7315 (16.0%) missing values	Missing
`source` has constant value "NASA"	Rejected

Pandas profiling

Variables：每个变量的统计信息

recclass
Categorical

Distinct count	466
Unique (%)	1.0%
Missing (%)	0.0%
Missing (n)	0



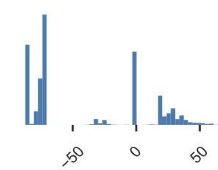
更详细的信息

Toggle details

reclat
Numeric

Distinct count	12739
Unique (%)	27.9%
Missing (%)	16.0%
Missing (n)	7315
Infinite (%)	0.0%
Infinite (n)	0

Mean	-39.10709514
Minimum	-87.36667
Maximum	81.16667
Zeros (%)	14.1%



Toggle details

强相关的会忽略并且提示

~~reclat_city~~
~~Highly correlated~~

This variable is highly correlated with `reclat` *and should be ignored for analysis*

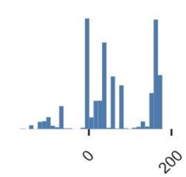
Correlation 0.9942518712

可视化展示

reclong
Numeric

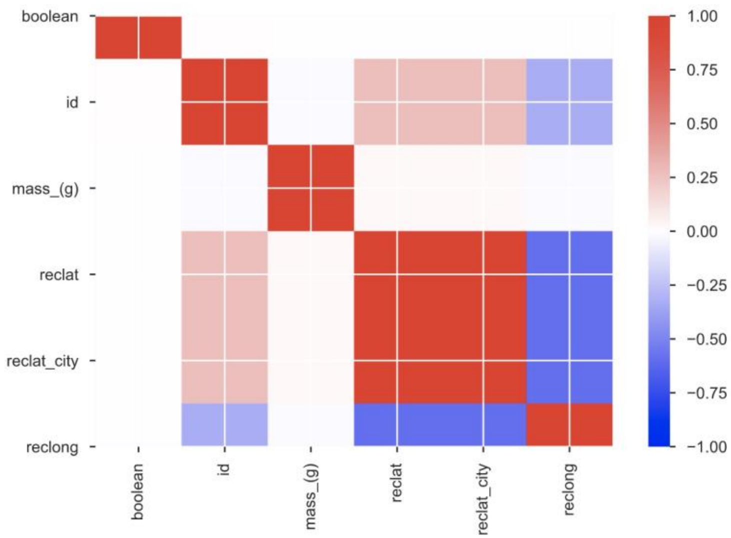
Distinct count	14641
Unique (%)	32.0%
Missing (%)	16.0%
Missing (n)	7315
Infinite (%)	0.0%
Infinite (n)	0

Mean	61.05259359
Minimum	-165.43333
Maximum	354.47333
Zeros (%)	13.6%



Pandas profiling

相关性



First rows

	boolean	fall	GeoLocation	id	mass_(g)	mixed	name	nametype	recclass	reclat
0	True	Fell	(50.775, 6.08333)	1	21.0	1	Aachen	Valid	L5	50.77500
1	False	Fell	(56.18333, 10.23333)	2	720.0	A	Aarhus	Valid	H6	56.18333
2	False	Fell	(54.21667, -113.0)	6	107000.0	1	Abee	Valid	EH4	54.21667
3	True	Fell	(16.88333, -99.9)	10	1914.0	A	Acapulco	Valid	Acapulcoite	16.88333
4	False	Fell	(-33.16667, -64.95)	370	780.0	1	Achiras	Valid	L6	-33.16667
5	False	Fell	(32.1, 71.8)	379	4239.0	A	Adhi Kot	Valid	EH4	32.10000
6	False	Fell	(44.83333, 95.16667)	390	910.0	A	Adzhi-Bogdo (stone)	Valid	LL3-6	44.83333
7	False	Fell	(44.21667, 0.61667)	392	30000.0	A	Agen	Valid	H5	44.21667
8	False	Fell	(-31.6, -65.23333)	398	1620.0	1	Aguada	Valid	L6	-31.60000
9	False	Fell	(-30.86667, -64.55)	417	1440.0	A	Aguila Blanca	Valid	L	-30.86667

前十行数据

2 Pandas under the hood

Python 数据科学栈

scikit learn

TensorFlow

PyTorch

mxnet

SciPy

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

NumPy

PYTHON 30th

Python vs NDArray

PyConChina 2019

- Python list

size

```
In [18]: import sys

In [19]: lst = list(range(1000))

In [20]: sys.getsizeof(lst)
Out[20]: 9112
```

speed

```
In [30]: lst = list(range(100000))

In [31]: %timeit sum((l + 1) for l in lst)
7.9 ms ± 128 µs per loop
```

- Numpy NDArray

size

```
In [24]: import numpy as np

In [25]: arr = np.arange(1000,
dtype=np.int32)

In [26]: sys.getsizeof(arr)
Out[26]: 4096
```

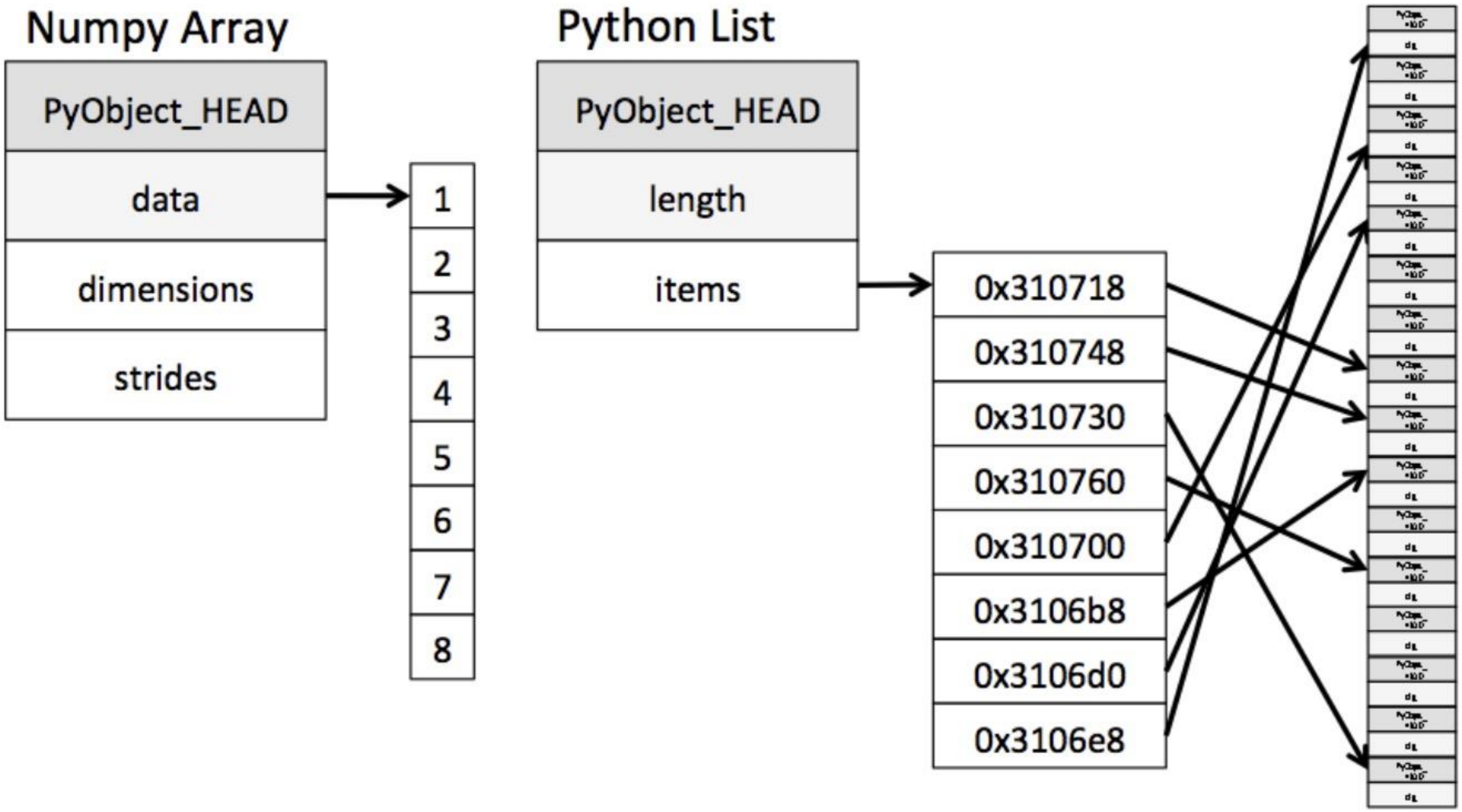
speed

```
In [32]: arr = np.arange(100000,
dtype=np.int32)

In [33]: %timeit (arr + 1).sum()
121 µs ± 2.32 µs per loop
```

PYTHON 30th

Python vs NDArray

PyConChina 2019



[Why python is slow]

PYTHON 30th

BlockManager

根据不同的数据类型分成不同的块（讲解一下 bytes）

DataFrame

	date	number_of_game	day_of_week	v_name	v_league	v_game_number	h_name	h_league	h_game_number	v_score	h_score	length_outs
0	01871054	0	Thu	CL1	na	1	FW1	na	1	0	2	54.0
1	18710505	0	Fri	BS1	na	1	WS3	na	1	20	18	54.0
2	18710506	0	Sat	CL1	na	2	RC1	na	1	12	4	54.0

IntBlock

	0	1	2	3	4	5
0	01871054	0	1	1	0	2
1	18710505	0	1	1	20	18
2	18710506	0	2	1	12	4

ObjectBlock

	0	1	2	3	4
0	Thu	CL1	na	FW1	na
1	Fri	BS1	na	WS3	na
2	Sat	CL1	na	RC1	na

FloatBlock

	0
0	54.0
1	54.0
2	54.0

https://www.dataquest.io/blog/pandas-big-data/

PYTHON 30th

BlockManager

	name	id	score
0	wang	0	90.0
1	zhao	1	99.9
2	he	2	60.0

```
In [6]: df._data.blocks
Out[6]:
(FloatBlock: slice(2, 3, 1), 1 x 3, dtype: float64,
 IntBlock: slice(1, 2, 1), 1 x 3, dtype: int64,
 ObjectBlock: slice(0, 1, 1), 1 x 3, dtype: object)

In [7]: type(df._data.blocks[1].values)
Out[7]: numpy.ndarray

In [8]: df._data.blocks[1].values
Out[8]: array([[0, 1, 2]])
```

DataFrame → BlockManager → Blocks

High level API | Translate | Storing data

3 更高效的 Pandas

任务：收集了县级城市的经纬度，计算他们与杭州的距离。

```
In [3]: df
Out[3]:
        行政代码          地名   longitude   latitude
0     130426          河北省涉县  113.742914  36.598105
1     130930  河北省孟村回族自治县  117.159538  38.091265
2     140728         山西省平遥县  112.265493  37.148090
3     150424         内蒙古林西县  118.110216  43.771462
4     210202      辽宁省大连市中山区  121.677966  38.900436
...      ...          ...         ...        ...
3516  621201      甘肃省陇南市市辖区  104.934573  33.394480
3517  621202      甘肃省陇南市武都区  105.134553  33.293917
3518  621221          甘肃省成县  105.688289  33.747297
3519  621222          甘肃省文县  104.784206  32.947265
3520  621223         甘肃省宕昌县  104.452827  34.013489

[3521 rows x 4 columns]
```

PyConChina 2019

haversine 公式:

```python
def haversine(lat1, lon1, lat2, lon2):
    # lat1, lon1 为位置 1 的经纬度坐标
    # lat2, lon2 为位置 2 的经纬度坐标
    import numpy as np

    dlon = np.radians(lon2 - lon1)
    dlat = np.radians(lat2 - lat1)
    a = np.sin(dlat / 2) ** 2 + np.cos(np.radians(lat1)) * \
        np.cos(np.radians(lat2)) * np.sin(dlon / 2) ** 2
    c = 2 * np.arcsin(np.sqrt(a))
    r = 6371  # 地球平均半径，单位为公里
    return c * r
```

```python
# 杭州的经纬度为 (120.2E ,30.3N)，计算距离调用如下
distance_hangzhou = haversine(lat, lon, 30.3, 120.2)
```

从循环开始说起

PyConChina 2019

```
In [8]: %%timeit
   ...:
   ...: haversine_distances = []
   ...: for _, row in df.iterrows():
   ...:     distance = haversine(row['latitude'], row['longitude'], 30.3, 120.2)
   ...:     haversine_distances.append(distance)
   ...:
```

```
471 ms ± 24.8 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Tip1：避免循环

Apply

```
In [12]: %%timeit
    ...:
    ...: haversine_distances = df.apply(lambda row: haversine(row['latitude'],
                                         row['longitude'], 30.3, 120.2), axis=1)
    ...:
```

```
146 ms ± 4.21 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

PyConChina 2019

加速你的 Pandas

```
In [12]: %%timeit
    ...:
    ...: haversine_distances = df.apply(lambda row: haversine(row['latitude'],
                                        row['longitude'], 30.3, 120.2), axis=1)
    ...:
146 ms ± 4.21 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Tip2：直接对列操作

精通面向数组的编程和思维方式是成为Python 科学计算牛人的一大关键步骤

—— Wes McKinney

```
In [19]: %%timeit
    ...:
    ...: df['dis'] = haversine(df['latitude'], df['longitude'], 30.3, 120.2)
    ...:
    ...:
```

```
2.62 ms ± 286 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

PYTHON 30th

加速你的 Pandas

还不够快？

```
In [20]: %%timeit
    ...:
    ...: df['dis'] = haversine(df['latitude'].to_numpy(),
                                df['longitude'].to_numpy(), 30.3, 120.2)
    ...:
    ...:
```

```
376 µs ± 7.55 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

```
471 ms ± 24.8 ms per loop
```

1000 倍以上的
提升

Tip3：数组计算直接使用 numpy

PYTHON 30th

分类数据

Categorical data

- 颜色： Red, Black, White, Green, …
- 性别： Male, Female
- 星期日期： Monday, Tuesday, Wednesday, …
- ……

绝大部分分类数据是字符串

Object 类型意味着大内存与低效

DataFrame

	date	number_of_game	day_of_week	v_name	v_league	v_game_number	h_name	h_league	h_game_number	v_score	h_score	length_outs
0	01871054	0	Thu	CL1	na	1	FW1	na	1	0	2	54.0
1	18710505	0	Fri	BS1	na	1	WS3	na	1	20	18	54.0
2	18710506	0	Sat	CL1	na	2	RC1	na	1	12	4	54.0

IntBlock

	0	1	2	3	4	5
0	01871054	0	1	1	0	2
1	18710505	0	1	1	20	18
2	18710506	0	2	1	12	4

ObjectBlock

	0	1	2	3	4
0	Thu	CL1	na	FW1	na
1	Fri	BS1	na	WS3	na
2	Sat	CL1	na	RC1	na

FloatBlock

	0
0	54.0
1	54.0
2	54.0

PYTHON 30th

分类数据


```
In [28]: df['省份']
Out[28]:
0          河北
1          河北
2          山西
         ...
3518       甘肃
3519       甘肃
3520       甘肃
Name: 省份, Length: 3521, dtype: object
```

用 category 表示省份

```
In [29]: df['省份'] = df['省份'].astype('category')

In [30]: df['省份']
Out[30]:
0       河北
1       河北
2       山西
        ..
3518    甘肃
3519    甘肃
3520    甘肃
Name: 省份, Length: 3521, dtype: category
Categories (33, object): [上海, 云南, 内蒙, 北京, ..., 陕西, 青海, 香港特别行政区, 黑龙江]
```

分类数据

PyConChina 2019

```
In [29]: series_as_category = df['省份']
Out[29]:
0       河北
1       河北
2       山西
        ..
3518    甘肃
3519    甘肃
3520    甘肃
Name: 省份, Length: 3521, dtype: category
Categories (33, object): [上海, 云南, 内蒙, 北京, ..., 陕西, 青海, 香港特别行政区, 黑龙江]
```

实际上已经映射成整型 (int8)

```
In [66]: series_as_category.cat.codes
Out[66]:
0       16
1       16
2       10
        ..
3518    23
3519    23
3520    23
Length: 3521, dtype: int8
```

分类数据

Tip4：使用 category 类型

- 内存节约 (~5.4 x)

```
In [49]: series = df['省份']

In [50]: series.memory_usage()
Out[50]: 28296

In [51]: series_as_category = series.astype('category')

In [52]: series_as_category.memory_usage()
Out[52]: 5193
```

- 性能 (~1.8 x)，数据量很大时性能会有大幅提升

```
In [58]: %timeit df.groupby('省份')['dis'].max()
913 µs ± 18.3 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

In [59]: %timeit df_category.groupby('省份')['dis'].max()
510 µs ± 15.8 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

当数据**规**模来到了 100G， 所有的**优**化都是白**费**,

求助 SQL? Spark?

不，我们需要并行的、分布式的 Pandas!

4 更大规模的数据分析

Mars

Mars: 一个统一的分布式计算框架，将 Python 科学技术栈应用到大数据中

- Numpy
- Pandas
- Scikit-learn

```
In [1]: import pandas as pd

In [2]: import numpy as np

In [3]: df =
pd.DataFrame(np.random.rand(10, 10))

In [4]: df.sum()
Out[4]:
0    4.639573
1    5.552392
2    4.974636
3    3.709493
4    3.886881
5    4.640418
6    5.660901
7    5.748306
8    4.227453
9    5.897225
dtype: float64
```

相同的 API

```
In [1]: import mars.tensor as mt

In [2]: import mars.dataframe as md

In [3]: df =
md.DataFrame(mt.random.rand(10, 10))

In [4]: df.sum().execute()
Out[4]:
0    4.657526
1    6.373730
2    5.526396
3    3.830616
4    5.070046
5    4.986147
6    5.586429
7    4.408413
8    4.461475
9    4.800761
dtype: float64
```

Mars

PyConChina 2019

```
In [5]: df1 = pd.DataFrame(np.random.rand(10, 3))

In [6]: df2 = pd.DataFrame(np.random.rand(10, 3),
index=[4, 1, 3, 2, 10, 5, 9, 8, 6, 7])

In [7]: df1 + df2
Out[7]:
           0         1         2
0        NaN       NaN       NaN
1   0.523616  0.877549  1.129148
2   1.420484  1.176361  1.504610
3   1.137251  0.582319  0.545606
4   1.433731  1.271197  1.069266
5   1.573455  1.078515  1.509255
6   0.916412  0.511708  0.741676
7   1.284630  0.579828  1.366145
8   1.283414  0.677225  0.136293
9   0.453015  1.801961  1.358312
10       NaN       NaN       NaN
```

```
In [5]: df1 = md.DataFrame(mt.random.rand(10, 3))

In [6]: df2 = md.DataFrame(mt.random.rand(10, 3),
index=[4, 1, 3, 2, 10, 5, 9, 8, 6, 7])

In [7]: (df1 + df2).execute()
Out[7]:
           0         1         2
0        NaN       NaN       NaN
1   0.342513  0.458722  1.106504
2   0.776831  0.549000  1.187729
3   0.483544  0.564384  0.245915
4   0.993470  0.102466  1.615469
5   0.744108  1.019650  1.522074
6   0.671610  0.820533  1.150278
7   0.826367  0.872842  0.877645
8   0.503570  1.328384  0.987532
9   1.654519  1.026768  1.456997
10       NaN       NaN       NaN
```

一致的行为，自动索引对齐

PYTHON 30th

Mars


```
In [10]: %%time
    ...: df = pd.DataFrame(np.random.rand(100000000, 4))
    ...: df.sum()
    ...:
    ...:
CPU times: user 12 s, sys: 3.6 s, total: 15.6 s
Wall time: 12.8 s
Out[10]:
0    4.999985e+07
1    4.999891e+07
2    4.999715e+07
3    5.000027e+07
dtype: float64
```

运行时间：12.8s

内存峰值：3430.29M

单机上，自动利用多核，
加速计算

```
In [4]: %%time
   ...: df = md.DataFrame(mt.random.rand(100000000, 4))
   ...: df.sum().execute()
   ...:
   ...:
CPU times: user 19.3 s, sys: 5.14 s, total: 24.4 s
Wall time: 4.27 s
Out[4]:
0    4.999827e+07
1    5.000004e+07
2    4.999853e+07
3    4.999748e+07
dtype: float64
```

运行时间：4.27s

内存峰值：2007.92M

Mars

分布式

部署 Mars 集群，通过 RESTful API 提交任务

```
In [5]: from mars.session import new_session

In [6]: sess = new_session('http://127.0.0.1:40002').as_default()

In [7]: df = md.DataFrame(mt.random.rand(100000000, 4))

In [8]: df.sum().execute()
```

PYTHON 30th

```
import numpy as np

def haversine(lat1, lon1, lat2, lon2):
    # lat1, lon1 为位置 1 的经纬度坐标
    # lat2, lon2 为位置 2 的经纬度坐标

    dlon = np.radians(lon2 - lon1)
    dlat = np.radians(lat2 - lat1)
    a = np.sin(dlat / 2) ** 2 +
np.cos(np.radians(lat1)) * \
        np.cos(np.radians(lat2)) * np.sin(dlon / 2)
** 2
    c = 2 * np.arcsin(np.sqrt(a))
    r = 6371  # 地球平均半径，单位为公里
    return c * r

%time haversine(latitude, longitude, 30.3, 120.2)

CPU times: user 11.5 s, sys: 9.6 s, total: 21 s
Wall time: 22 s
```

```
import mars.tensor as np

def haversine(lat1, lon1, lat2, lon2):
    # lat1, lon1 为位置 1 的经纬度坐标
    # lat2, lon2 为位置 2 的经纬度坐标

    dlon = np.radians(lon2 - lon1)
    dlat = np.radians(lat2 - lat1)
    a = np.sin(dlat / 2) ** 2 +
np.cos(np.radians(lat1)) * \
        np.cos(np.radians(lat2)) * np.sin(dlon / 2)
** 2
    c = 2 * np.arcsin(np.sqrt(a))
    r = 6371  # 地球平均半径，单位为公里
    return (c * r).execute()

%time haversine(latitude, longitude, 30.3, 120.2)
CPU times: user 20.4 s, sys: 19.2 s, total: 39.5 s
Wall time: 13.2 s
```

Tip5：使用 Mars

如何并行计算

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
0	2	3	6	0	5	2	2	6	9	3
1	5	7	7	8	3	5	4	8	2	8
2	0	7	2	8	5	4	0	7	4	5
3	5	5	7	1	8	9	7	6	2	2
4	2	0	2	4	7	6	6	2	0	9
5	6	7	6	9	4	8	5	7	6	2
6	9	7	2	6	4	3	4	9	1	0
7	7	0	0	0	8	4	6	0	2	1
8	0	7	2	1	6	2	6	2	5	1
9	0	7	6	1	9	5	9	2	4	9

分块求和

c0	14
c1	22
c2	24
c3	21
c4	28

c0	22
c1	28
c2	16
c3	17
c4	31

c5	26
c6	19
c7	29
c8	17
c9	27

c5	22
c6	30
c7	20
c8	18
c9	13

组合

	0	1
c0	14	22
c1	22	28
c2	24	16
c3	21	17
c4	28	31
c5	26	22
c6	19	30
c7	29	20
c8	17	18
c9	27	13

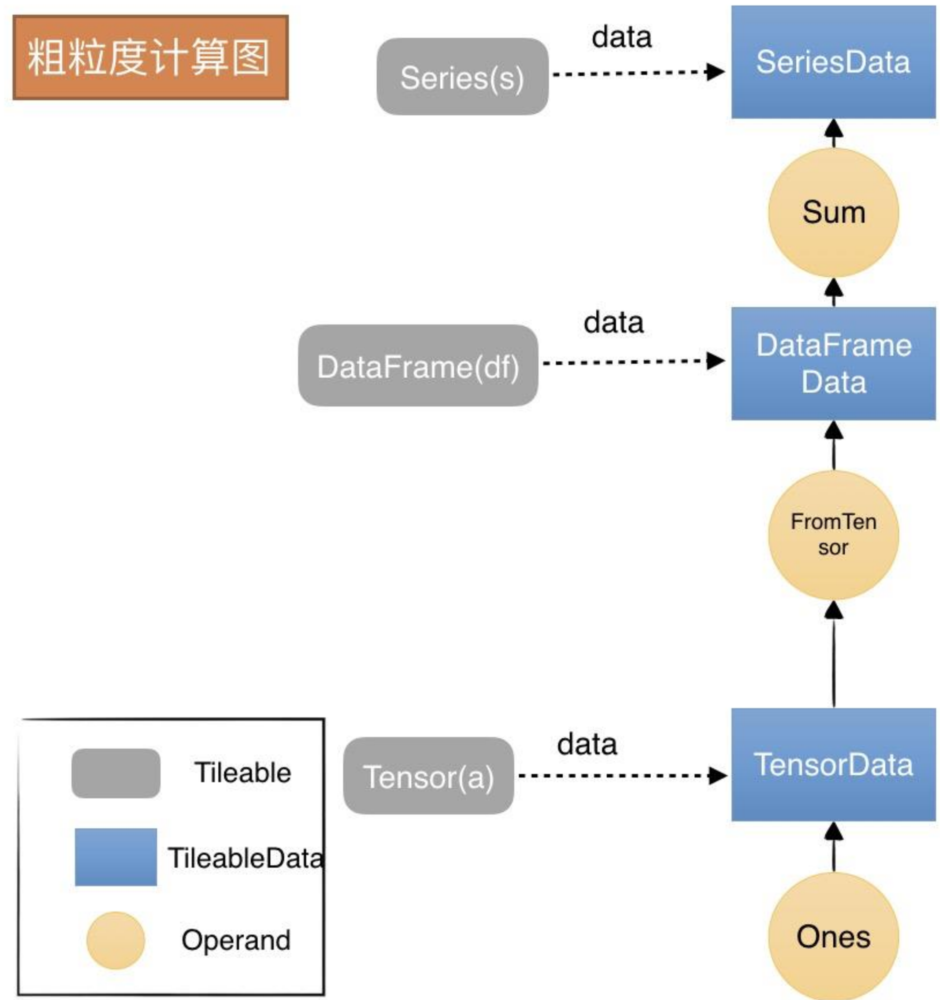
求和

c0	36
c1	50
c2	40
c3	38
c4	59
c5	48
c6	49
c7	49
c8	35
c9	40

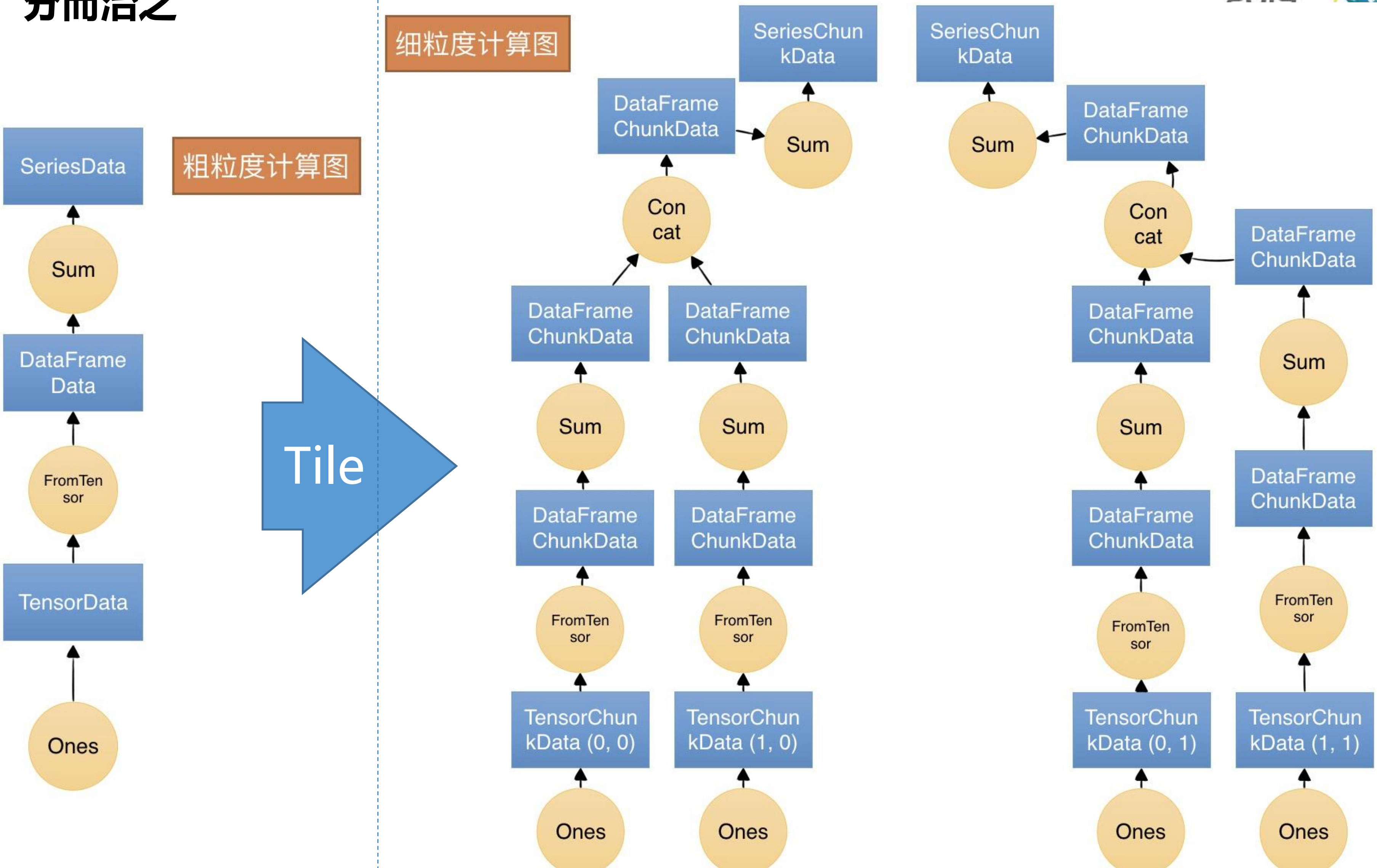
如何并行计算

分而治之

```
In [1]: import mars.tensor as mt

In [2]: import mars.dataframe as md

In [3]: a = mt.ones((10, 10),
chunk_size=5)

In [4]: df = md.DataFrame(a)

In [5]: s = df.sum()

In [6]: s.execute()
Out[6]:
0    10.0
1    10.0
2    10.0
3    10.0
4    10.0
5    10.0
6    10.0
7    10.0
8    10.0
9    10.0
dtype: float64
```

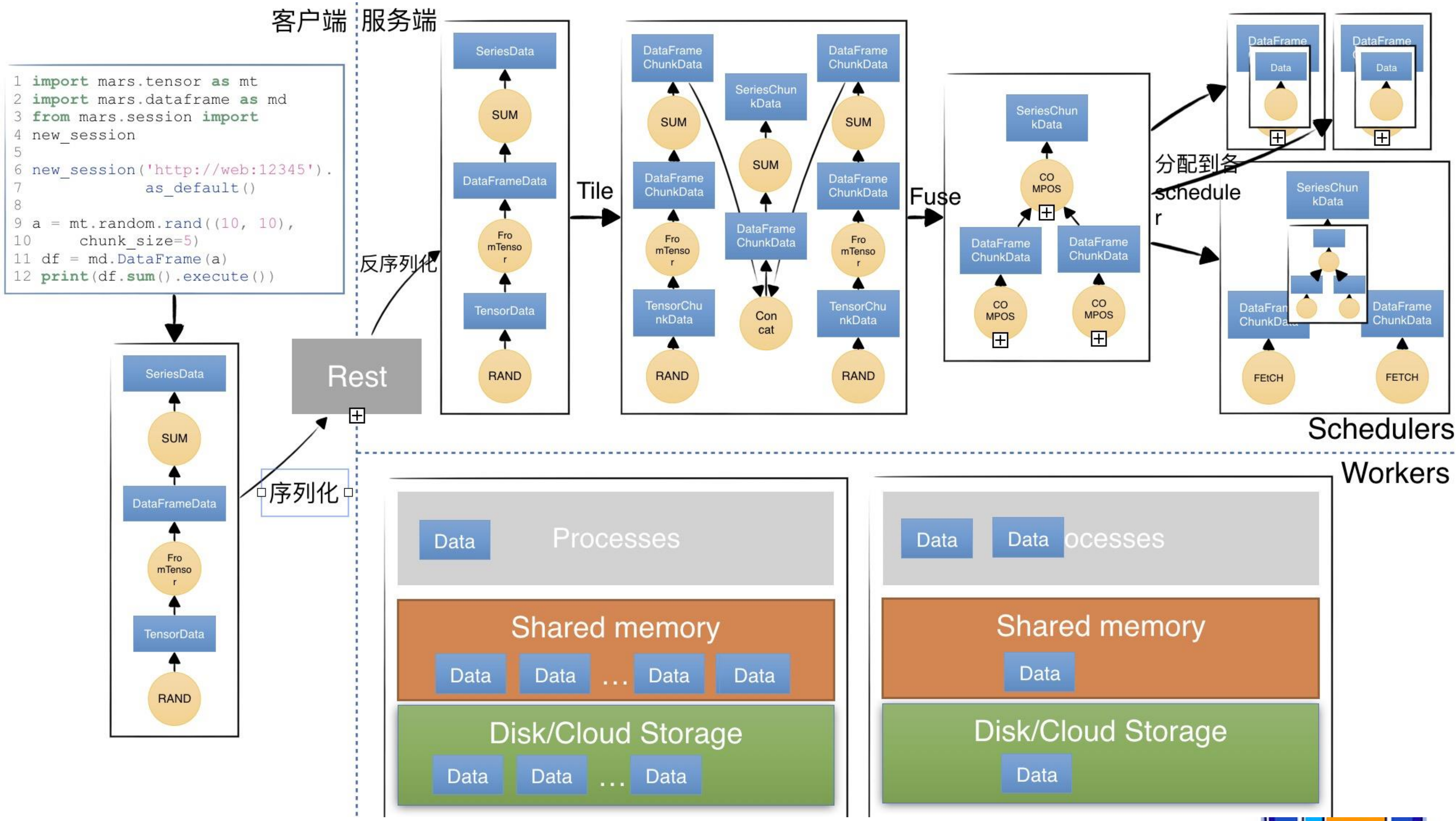


分而治之



粗粒度计算图
SeriesData
Sum
DataFrame Data
FromTensor
TensorData
Ones
Tile
细粒度计算图
SeriesChunkData
Sum
DataFrame ChunkData
Concat
DataFrame ChunkData
Sum
DataFrame ChunkData
FromTensor
TensorChunkData (0, 0)
TensorChunkData (1, 0)
TensorChunkData (0, 1)
TensorChunkData (1, 1)
Ones

分布式执行


客户端
服务端
import mars.tensor as mt
import mars.dataframe as md
from mars.session import new_session
new_session('http://web:12345').as_default()
a = mt.random.rand((10, 10), chunk_size=5)
df = md.DataFrame(a)
print(df.sum().execute())
SeriesData
SUM
DataFrameData
FromTensor
TensorData
RAND
序列化
Rest
反序列化
Tile
DataFrame ChunkData
SeriesChunkData
TensorChunkData
Concat
Fuse
COMPOS
分配到各scheduler
Data
FEtCH
FETCH
Schedulers
Workers
Data
Processes
Shared memory
Data ... Data
Disk/Cloud Storage
Data ... Data

Mars DataFrame

- 实现的接口
  - 创建 DataFrame：DataFrame、from_records
  - Basic arithmetic：基本算数运算
  - Math：数学运算
  - Indexing: 索引
    - iloc
    - 列选择
    - set_Index
  - Reduction：聚合
  - Groupby：分组聚合
  - merge/join
- Roadmap：https://github.com/mars-project/mars/issues/495

Mars 现状及方向

- pip install pymars
- Mars 开源地址：https://github.com/mars-project/mars
- Mars 文档：https://docs.mars-project.io/zh_CN/latest/
- 双版本发布
- 方向：
  - 社区是重点
  - 技术：
    - Roadmap 和 Enhancement Proposal：https://github.com/mars-project/mars/issues/537
    - 丰富 DataFrame、 learn 和 Tensor 的接口
    - 更好的算子融合
    - Mars actors 层优化，支持更高效的执行和网络效率
    - 支持更多调度

THANK YOU

Pole

浙江 杭州