# Key Term

- **numpy.ndarray** - n-dimensional Numpy array, the core object that underpins Pandas DataFrames
- **pyspark.sql.DataFrame** - Distributed dataframe object in PySpark for working with large datasets
- **dask.dataframe.DataFrame** - Dask dataframe that partitions Pandas dataframe across multiple cores
- **np.random.randn()** - Generate an ndarray with random values from a normal distribution
- **df.persist()** - Trigger caching of a PySpark dataframe into memory or disk storage

## Pandas Code Examples

```python
import pandas as pd
import numpy as np

# numpy.ndarray
data = np.random.randn(5, 4)  # Generate 5x4 ndarray
df = pd.DataFrame(data) # Underlying structure of Pandas DataFrame
print(df)

# np.random.randn()
df = pd.DataFrame(np.random.randn(5, 4),
                  columns=['A', 'B', 'C', 'D'])
print(df)
```

```
          0         1         2         3
0 -0.030742  0.477710  0.029938 -1.005135
1 -0.137934  0.299512 -1.404028 -0.153353
2 -0.755672 -0.451716 -0.182981  2.478212
3 -0.089476  0.472844  0.329931 -0.630641
4 -1.144661 -0.177849 -1.287844 -1.102989
          A         B         C         D
0  0.465819 -1.195653  0.618216  0.483000
1 -1.423888  0.671257 -1.298546 -1.954487
2  0.180366 -2.034974 -0.939537 -0.191529
3 -0.434719  0.248863 -2.398414 -0.791016
4 -0.191686 -0.278751  0.525708 -2.467836
```

# Creating NumPy Arrays in Python

## NumPy

- Designed for Scientific Computing
- Ndarray object = stand for N-dimensional array object → refer to as an array
- Highly optimized computational tools
- Ecosystem of Scientific and Mathematical Libraries

↳ Machine Learning: There are many the machine learning libraries that expect an array object as input
↳ if they don't, they will turn the input into an array object behind the scenes and the return an array object as a result

## ndarray

- Size set at creation (its size is set; cannot resize it without creating a new object)
- Single data type
- Not limited to two dimensions
- Can be reshaped returning a new array object

## NumPy Arrays

```
[65]: import numpy as np
```

### Creating Arrays

```
[66]: data = [1, 2, 3, 4]
      first_array = np.array(data)
      first_array
[66]: array([1, 2, 3, 4])
```
↓be an array with that list

```
[67]: data = [[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]]
      my_array = np.array(data)
      my_array
[67]: array([[1, 2, 3],
             [4, 5, 6],
             [7, 8, 9]])
```
↓be an array with the list of lists

## Creating

```
[69]: np.ones(12)
[69]: array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```
↳ make an array 1 twelve times

```
[70]: np.zeros(3)
[70]: array([0., 0., 0.])
```
↓ also zeros

```
[71]: np.arange(10)
[71]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```
↳ create an array with list of range

```
[72]: np.arange(3, 12, 4)
[72]: array([ 3,  7, 11])
```
Starting number; Ending number; ↳ step

## Dimensions and Shape

```
[73]: oned = np.arange(21)
      oned
[73]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
             17, 18, 19, 20])
[74]: oned.shape
[74]: (21,)
```
cause it ↓ 1D

```
[75]: oned.ndim
[75]: 1
```
→ check its number of dimensions

```
[76]: oned.size
[76]: 21
```

```
[77]: list_o_lists = [[1,2,3], [4,5,6], [7,8,9]]
      twod = np.array(list_o_lists)
      twod
[77]: array([[1, 2, 3],
             [4, 5, 6],
             [7, 8, 9]])
```
↳ list of lists; so it is 2D

```
[78]: twod.ndim
[78]: 2
[79]: twod.size
[79]: 9
[80]: twod.shape
[80]: (3, 3)
[81]: oned = np.arange(12)
      oned
[81]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

3×4 = 12 correct (4 = column, 3 = rows)

```
[82]: twod = oned.reshape(3,4)
      twod
[82]: array([[ 0,  1,  2,  3],
             [ 4,  5,  6,  7],
             [ 8,  9, 10, 11]])
```
↳ use reshape method to return array with different shape

```
[83]: twod.shape
[83]: (3, 4)
[84]: twod.ndim
[84]: 2
```
from 1D to 2D

incorrect: 2×3=6

```
[85]: twod.reshape(2,3)
```
to use reshape, those arguments must be numbers that will add up to the right number of times in your array

```
ValueError                                Traceback (most recent call last)
/var/folders/29/5dyw2t0n71v13sp1980c30g80000gr/T/ipykernel_30144/198994001.py in <module>
----> 1 twod.reshape(2,3)

ValueError: cannot reshape array of size 12 into shape (2,3)
```
it's like how TF to turn shape 12 to shape 6

2×2×3 = 12 correct

```
[86]: twod.reshape(2,2,3)
[86]: array([[[ 0,  1,  2],
              [ 3,  4,  5]],

             [[ 6,  7,  8],
              [ 9, 10, 11]]])
```

# Cont. Creating NumPy Arrays in Python

## setting Data Type

### Setting Data Type → one of the things that separates Numpy from Pandas, is that they can only contain a single data type

```
[87]: darray = np.arange(100)
      darray

[87]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
             17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
             34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
             51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
             68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
             85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])

[88]: darray.dtype
      ↓ check its type
[88]: dtype('int64')
      ↓ can also check its size
[89]: darray.nbytes

[89]: 800
```

```
[90]: darray = np.arange(100, dtype=np.int8)
      darray

[90]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
             17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
             34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
             51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
             68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
             85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99],
            dtype=int8)

[91]: darray.dtype

[91]: dtype('int8')

[92]: darray.nbytes

[92]: 100
```

↑ we can add new argument to specify a particular data type

decrease memory

↳ we can fit them to small memory

- Once you set its datatype, NumPy will not let you add items of a different data type

```
[93]: darray[12] = 'a'

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/var/folders/29/5dyw2t0n71v13sp1980c30g80000gr/T/ipykernel_30144/1692319446.py in <module>
----> 1 darray[12] = 'a'

ValueError: invalid literal for int() with base 10: 'a'

[94]: darray[12] = 0.4
[95]: darray[12]
[95]: 0
```

[94]: its data type is the int8 but this value is decimal

[95]: so it stripped off

↳ so be aware of data type, when using Numpy array

## Broadcasting

### Broadcasting

```
[96]: A1 = np.array([[1,2,3],
                     [4,5,6],
                     [7,8,9]])

[97]: A1.shape

[97]: (3, 3)

[ ]:

[98]: A2 = np.array([[1,1,1],
                     [1,1,1],
                     [1,1,1]])

[99]: A1 + A2
[99]: array([[ 2,  3,  4],
             [ 5,  6,  7],
             [ 8,  9, 10]])

[100]: A1 + 3
[100]: array([[ 4,  5,  6],
              [ 7,  8,  9],
              [10, 11, 12]])
```

[99]: values within the arrays are added together

[100]: we can Add a single value to the array

```
[101]: A3 = np.array([1,1,1])

[102]: A3.shape

[102]: (3,)

[103]: A1 + A3
[103]: array([[ 2,  3,  4],
              [ 5,  6,  7],
              [ 8,  9, 10]])

[104]: A4 = np.arange(10).reshape(2,1,5)
       A4

[104]: array([[[0, 1, 2, 3, 4]],

              [[5, 6, 7, 8, 9]]])

[105]: A5 = np.arange(14).reshape(2,7,1)
       A5

[105]: array([[[ 0],
               [ 1],
               [ 2],
               [ 3],
               [ 4],
               [ 5],
               [ 6]],

              [[ 7],
               [ 8],
               [ 9],
               [10],
               [11],
               [12],
               [13]]])
```

expanded out, it chose the maximum dimensions of the input arrays

its still added to every single number

and be a 3 by 3 array

This is called Broadcasting

```
[106]: A6 = A4 + A5

[107]: A6.shape

[107]: (2, 7, 5)
```

Notice it selected by maximum size from each dimension

# Cont. Creating NumPy Arrays in Python

## Matrix Operations

```
[113]: M1 = np.arange(9).reshape(3,3)
       M1

[113]: array([[0, 1, 2],
              [3, 4, 5],
              [6, 7, 8]])

[116]: M2 = np.arange(2, 11).reshape(3,3)
       M2

[116]: array([[ 2,  3,  4],
              [ 5,  6,  7],
              [ 8,  9, 10]])

[117]: M1.transpose()

[117]: array([[0, 3, 6],
              [1, 4, 7],
              [2, 5, 8]])

[118]: M1.diagonal()

[118]: array([0, 4, 8])

[119]: M1 @ M2

[119]: array([[ 21,  24,  27],
              [ 66,  78,  90],
              [111, 132, 153]])
```

matrix product

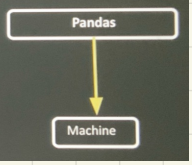
## When to use them

| ndarray | DataFrame |
| --- | --- |
| Multi dimensional data (any dim that's not 2 dim) | Two dimensional data |
| Single data type | Multiple data types |
| Complex numerical calculations | Data analysis |
| | Data visualization |

# Spark and PySpark DataFrames in Python

## Pandas

- Designed to run on single machine
- Performance bound by machines memory
- Chunking ↳ load a subset of data and work on that and then load another chunk of data
- Upper limit Gigabytes (1? 5? 100?)
  ↳ generally



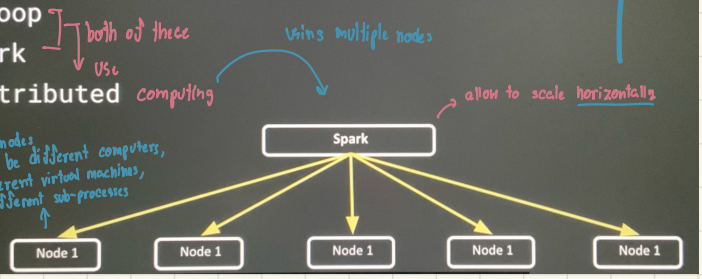

## Big Data → Terrabytes, Petabytes

- Hadoop
- Spark
- Distributed computing

both of these use

uses multiple nodes

allow to scale horizontally




these nodes could be different computers, different virtual machines, or different sub-processes

- add machines to increase performance not necessarily have to add much larger and more expensive machines

Hadoop:
- does by writing intermediate files during calculations
  ↳ doesn't require a great amount of RAM

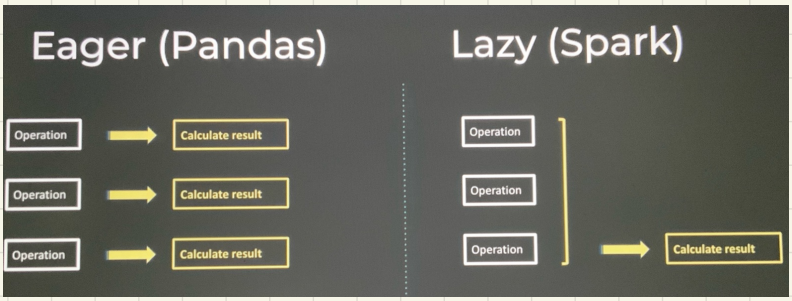
Spark:
- newer solution
- more performant, does work in memory
- by using memory all nodes

## Spark

Java virtual machine

- Distributed DataFrames on JVM
- Written in Scala
- PySpark Library ← include, use spark dataframe by using Python context
- Data sources include Hadoop HDFS, S3, and Streaming
- Lazy Evaluation

- its core uses DataFrames ← Distributed data frame




↳ every operation in Pandas code calculates its result before the next operation is run
↳ useful when interactively, intuitive to debug

↳ taking advantage of PySpark
↳ Operations are stacked and optimized transformation is calculated by Spark ← run in background
↳ use its nature distributed

## Considerations between Pandas or Spark

- Built for big data transformations
  ↳ more than Gb, use spark
  ↳ <= Gb like csv file, use Pandas
- Optimized integration plan strategy
  ↳ are you plan to the way you write your transformations? Eager or Lazy?
- Debugging challenges
  ↳ Spark is challenging to do a debugging sometime

# Creating Dask DataFrames in Python

## Dask

- Python native (it's a library)
- Distributed operations (like spark)
- Pandas DataFrames (wraps)
- NumPy arrays
- Arbitrary Python code

## Dask

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

[4]: leng = 100000000  ← large now
     data = { 'a': (random.randint(0, 100) for _ in range(leng)),
              'b': (random.randint(2, 200) for _ in range(leng))}

[5]: df = pd.DataFrame(data)
     df.head()

[5]:    a    b
     0  94  130
     1   0  167
     2  41   40
     3  97  146
     4  46    6

[10]: df.std()

[10]: a    29.153448
      b    57.449703
      dtype: float64
```

before complete this, it took time for a while

## Dask

```
[11]: import dask.dataframe as dd

[12]: ddf = dd.from_pandas(df, npartitions=3)

[13]: ddf

[13]: Dask DataFrame Structure:
                      a      b
      npartitions=3
                      0  int64  int64
      33333334      ...    ...
      66666668      ...    ...
      99999999      ...    ...
      Dask Name: from_pandas, 3 tasks
```

method · data · num. that we wanted to divide the data

no actual values

```
[14]: ddf.std()

[14]: Dask Series Structure:
      npartitions=1
      a    float64
      b        ...
      dtype: float64
      Dask Name: dataframe-std, 15 tasks

[15]: ddf.std().compute()

[15]: a    29.153448
      b    57.449703
      dtype: float64
```

→ it didn't give the answer

→ because it uses lazy evaluation like spark

← to force Dask to execute

```
[16]: result = ddf.a.sum() - ddf.b.sum()
      result

[16]: dd.Scalar<sub-0ad..., dtype=int64>

[17]: result.compute()

[17]: -5099626371
```

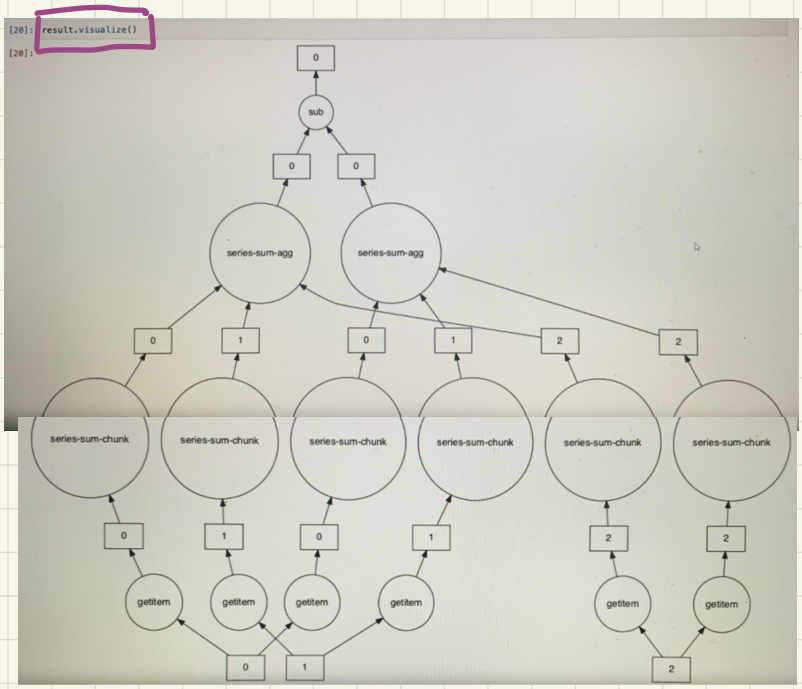
```
[18]: result.dask

[18]: HighLevelGraph
      HighLevelGraph with 8 layers.
      ▸ Layer1: from_pandas
      ▸ Layer2: getitem
      ▸ Layer3: series-sum-chunk
      ▸ Layer4: series-sum-agg
      ▸ Layer5: getitem
      ▸ Layer6: series-sum-chunk
      ▸ Layer7: series-sum-agg
      ▸ Layer8: sub
```

It creates a graph behind the scenes

various mechanism for calculating

The command

```
[20]: result.visualize()
```




## Choosing a framework

| | | |
|---|---|---|
| Pandas or NumPy | Vs | PySpark or Dask |
| | Size of data (small-medium / large) | |
| Pandas | Vs | NumPy |
| | Nature of data (visualization, 2 dim, various data type / more dim, single data type) | |
| PySpark | Vs | Dask |
| | Nature of enterprise (large ecosystem / Python native, all pipeline with Python) | |