Pierre Navaro

Python

tools for big data

UNIVERSITÉ RENNES 2

Publisher

Contents

Chapter 1

Git

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Official website https://git-scm.com

1.1 GitHub

- Web-based hosting service for version control using Git.
- Offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features.
- Provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.
- Github is the largest host of source code in the world.

1.2 About SCM

- Records changes to a file or set of files over time.
- You can recall specific versions later.
- You can use it with nearly any type of file on a computer.
- This is the better way to collaborate on the same document.
- Every change is committed with an author and a date.
- Figures are downloaded from Pro Git book.
- ”Become a git guru” tutorial (https://www.atlassian.com/git/tutorials).

1.3 Local Version Control System

- Collaboration is not really possible.

1.4 Distributed Version Control Systems

- Clients fully mirror the repository.
- You can collaborate with different groups of people in different ways simultaneously within the same project.
- No need of network connection.
- Multiple backups.

1.5 Configure Git

Settings are saved on the computer for all your git repositories.

```
git config --global user.name "Prenom Nom"
git config --global user.email "prenom.nom@univ-rennes2.fr"
```

1.6 Cloning the repository

```
git clone ssh://svmass2/git/big-data.git
```

or

```
git clone https://github.com/pnavaro/big-data.git
```

To save your work locally create a branch

```
git checkout -b my_branch
```

1.7 Four File status in the repository

1.8 Git Workflow

1.9 Locally saving your modifications

Get your files status

```
In [1]: %%bash
        git status
```

```
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>…" to include in what will be committed)
        ../.doit.db.dat
        ../spark-3.0.0-bin-hadoop2.7.tgz
        ../spark-3.0.0-bin-hadoop2.7/

nothing added to commit but untracked files present (use "git add" to track)
```

1.10 Add the modified file to the index

```
git add your_notebook_copy.ipynb
```

Checking which files are ready to be committed.

```
In [2]: %%bash
        git status
```

```
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        ../.doit.db.dat
        ../spark-3.0.0-bin-hadoop2.7.tgz
        ../spark-3.0.0-bin-hadoop2.7/

nothing added to commit but untracked files present (use "git add" to track)
```

Now save your work, the branch is local.

```
git commit -m 'Some modifications'
```

1.11 Updating from the Repository

If the master branch has changed. To get all new updates :

```
In [3]: %%bash
        git pull origin master
```

```
Already up to date.


warning: Pulling without specifying how to reconcile divergent branches is
discouraged. You can squelch this message by running one of the following
commands sometime before your next pull:

  git config pull.rebase false  # merge (the default strategy)
  git config pull.rebase true   # rebase
  git config pull.ff only       # fast-forward only

You can replace "git config" with "git config --global" to set a default
preference for all repositories. You can also pass --rebase, --no-rebase,
or --ff-only on the command line to override the configured default per
invocation.

From https://github.com/pnavaro/big-data
 * branch            master     -> FETCH_HEAD
```

1.12 Solve conflicts

- If you have some conflicts, no problem just do :

```
git checkout notebook.ipynb
```

It will give you back the original version of notebook.ipynb

- If you have big troubles, you can do

```
git reset --hard
```

Be careful with this last command, you remove uncommited changes.

1.13 Git through IDE

- Install bash-completion and source git-prompt.sh.
- Use Gui tools:
  - GitHub Desktop
  - Sourcetree
  - GitKraken
- VCS plugin of IDE
  - RStudio
  - Eclipse
  - JetBrains

Chapter 2

Installation with conda

2.1 Install Anaconda (large) or Miniconda (small)

2.2 Open a terminal (Linux/MacOSX) or a Anaconda prompt (Windows)

2.3 Create a new conda environment from file

```
cd big-data
conda env create
```

Conda envs documentation.

2.4 Activate the new environment (for all students)

Activating the conda environment will change your shell's prompt to show what virtual environment you're using, and modify the environment so that running python will get you that particular version and installation of Python.

You must do this everytime you open a new terminal

2.5 Install the kernel for jupyter

```
python -m ipykernel install --user \
       --name big-data
```

Chapter 3

Jupyter

3.1 Launch Jupyter server

```
jupyter notebook
```

- Go to notebooks folder
- Open the file 03.JupyterQuickStart.ipynb

3.2 Make a Copy

Before modifying the notebook, make a copy of it. Go to to `File` menu on the top left of the notebook and click on `Make a Copy...`

3.3 Jupyter Notebook

Jupyter notebook, formerly known as the IPython notebook, is a flexible tool that helps you create readable analyses, as you can keep code, images, comments, formulae and plots together.

Jupyter is quite extensible, supports many programming languages and is easily hosted on your computer or on almost any server — you only need to have ssh or http access. Best of all, it's completely free.

The name Jupyter is an indirect acronyum of the three core languages it was designed for: **JU**lia, **PYT**hon, and **R**

3.4 Keyboard Shortcuts

- To access keyboard shortcuts, use the command palette: `Cmd + Shift + P`
- `Esc` will take you into command mode where you can navigate around your notebook with arrow keys.
- While in command mode:
- A to insert a new cell above the current cell, B to insert a new cell below.
- M to change the current cell to Markdown, Y to change it back to code
- D + D (press the key twice) to delete the current cell

3.5 Easy links to documentation

- Shift + Tab will also show you the Docstring

```
In [1]: dict
```

```
Out[1]: dict
```

3.6 Magic commands

```
In [2]: %lsmagic
```

```
Out[2]: Available line magics:
        %alias  %alias_magic  %autoawait  %autocall  %automagic  %autosave  %bookmark  %cat  %cd  %clea

        Available cell magics:
        %%!  %%HTML  %%SVG  %%bash  %%capture  %%debug  %%file  %%html  %%javascript  %%js  %%latex  %%

        Automagic is ON, % prefix IS NOT needed for line magics.
```

```
In [3]: %ls
```

```
01-GitBasics.ipynb                book-018.dat  book-056.dat  book-094.dat
02-Installation.ipynb             book-019.dat  book-057.dat  book-095.dat
03-JupyterQuickStart.ipynb        book-020.dat  book-058.dat  book-096.dat
04-WordCount.ipynb                book-021.dat  book-059.dat  book-097.dat
05-MapReduce.ipynb                book-022.dat  book-060.dat  book-098.dat
06-ParallelComputation.ipynb      book-023.dat  book-061.dat  book-099.dat
07-AsynchronousProcessing.ipynb   book-024.dat  book-062.dat  data/
08-DaskDelayed.ipynb              book-025.dat  book-063.dat  fibonacci.py
09-DaskBag.ipynb                  book-026.dat  book-064.dat  images/
10-PandasSeries.ipynb             book-027.dat  book-065.dat  log
11-PandaDataframes.ipynb          book-028.dat  book-066.dat  mapper.py*
12-PySpark.ipynb                  book-029.dat  book-067.dat  mydask.png
13-UnixCommands.ipynb             book-030.dat  book-068.dat  pmap.py
14-Hadoop.ipynb                   book-031.dat  book-069.dat  reducer.py*
15-HadoopFileFormats.ipynb        book-032.dat  book-070.dat  sample00.txt
16-DaskDataframes.ipynb           book-033.dat  book-071.dat  sample01.txt
17-SparkDataFrames.ipynb          book-034.dat  book-072.dat  sample02.txt
18-NYCTaxiCabTripDask.ipynb       book-035.dat  book-073.dat  sample03.txt
19-NYCTaxiCabTripSpark.ipynb      book-036.dat  book-074.dat  sample04.txt
Makefile                          book-037.dat  book-075.dat  sample05.txt
```

```
book-000.dat                    book-038.dat  book-076.dat  sample06.txt
book-001.dat                    book-039.dat  book-077.dat  sample07.txt
book-002.dat                    book-040.dat  book-078.dat  sample08.txt
book-003.dat                    book-041.dat  book-079.dat  sample09.txt
book-004.dat                    book-042.dat  book-080.dat  sample10.txt
book-005.dat                    book-043.dat  book-081.dat  sample11.txt
book-006.dat                    book-044.dat  book-082.dat  sample12.txt
book-007.dat                    book-045.dat  book-083.dat  sample13.txt
book-008.dat                    book-046.dat  book-084.dat  sample14.txt
book-009.dat                    book-047.dat  book-085.dat  sample15.txt
book-010.dat                    book-048.dat  book-086.dat  sample16.txt
book-011.dat                    book-049.dat  book-087.dat  sample17.txt
book-012.dat                    book-050.dat  book-088.dat  sample18.txt
book-013.dat                    book-051.dat  book-089.dat  sample19.txt
book-014.dat                    book-052.dat  book-090.dat  test_stdin.py
book-015.dat                    book-053.dat  book-091.dat
book-016.dat                    book-054.dat  book-092.dat
book-017.dat                    book-055.dat  book-093.dat
```

```
In [4]: %%file sample.txt

        write the cell content to the file sample.txt.
        The file is created when you run this cell.
```

```
Writing sample.txt
```

```
In [5]: %cat sample.txt
```

```
write the cell content to the file sample.txt.
The file is created when you run this cell.
```

```
In [6]: %%file fibonacci.py

        f1, f2 = 1, 1
        for n in range(10):
            print(f1, end=',')
            f1, f2 = f2, f1+f2
```

```
Overwriting fibonacci.py
```

```
In [7]: %run fibonacci.py
```

```
1,1,2,3,5,8,13,21,34,55,
```

```
In [8]: # %load fibonacci.py

        f1, f2 = 1, 1
        for n in range(10):
            print(f1, end=',')
            f1, f2 = f2, f1+f2
```

```
1,1,2,3,5,8,13,21,34,55,
```

```
In [9]: %%time
        f1, f2 = 1, 1
        for n in range(10):
            print(f1, end=',')
            f1, f2 = f2, f1+f2
        print()
```

```
1,1,2,3,5,8,13,21,34,55,
CPU times: user 311 µs, sys: 97 µs, total: 408 µs
Wall time: 267 µs
```

3.7 Installing Python Packages from a Jupyter Notebook

3.7.1 Install a conda package in the current Jupyter kernel

Example with package `numpy` from *conda-forge*

```
%conda install -c conda-forge lorem
```

Chapter 4

Wordcount

- Wikipedia
- Word count example reads text files and counts how often words occur.
- Word count is commonly used by translators to determine the price for the translation job.
- This is the "Hello World" program of Big Data.

Some recommendations: - *Don't google too much, ask me or use the python documentation through* `help` *function. - Do not try to find a clever or optimized solution, do something that works before. - Please don't get the solution from your colleagues* - *Notebooks will be updated next week with solutions*

4.1 Create sample text file

```
In [1]: from lorem import text

        with open("sample.txt", "w") as f:
            for i in range(10000):
                f.write(text())
```

4.1.1 Exercise 4.1

Write a python program that counts the number of lines, words and characters in that file.

```
In [2]: %%bash
        wc sample.txt
        du -h sample.txt

   70234  2019444 14267810 sample.txt
14M        sample.txt
```

- Compute number of lines

```
In [3]: with open("sample.txt") as f:
            lines = list(f)

        nlines = len(lines)
        nlines

Out[3]: 70235
```

- Compute number of words

```
In [4]: nwords = sum([len(line.split()) for line in lines])
        nwords

Out[4]: 2019444

In [5]: nchars = 0
        for line in lines:
            words = line.split()
            nchars += sum([len(word) for word in line.split()])

        nchars

Out[5]: 12213250
```

- `set` gives the list of unique elements from words list.

```
In [6]: s = set(words)
        s

Out[6]: {'Dolore',
          'Dolorem',
          'Eius',
          'Modi',
          'Numquam',
          'Quaerat',
          'Sed',
          'adipisci',
          'adipisci.',
          'aliquam',
          'amet',
          'consectetur',
          'dolor.',
          'dolore',
          'dolorem',
          'dolorem.',
          'eius',
          'eius.',
          'est',
          'etincidunt',
          'ipsum.',
          'labore',
          'magnam',
          'modi',
          'neque',
          'non',
          'quaerat',
          'quisquam',
          'sed',
          'sit',
          'sit.',
          'tempora',
          'ut',
          'ut.',
          'velit',
          'voluptatem',
          'voluptatem.'}
```

4.1.2 Exercise 4.2

Create a function called `map_words` that take a file name as argument and return a lists containing all words as items.

```
map_words("sample.txt")[:5] # first five words
['adipisci', 'adipisci', 'adipisci', 'adipisci', 'adipisci']
```

```
In [7]: def map_words(filename):
            """ take a file name as argument and return a
            lists containing all words as item
            """
            with open(filename) as f:
                data = f.read().lower().replace('.',' ')

            return sorted(data.split())

        map_words("sample.txt")[:5]
```

```
Out[7]: ['adipisci', 'adipisci', 'adipisci', 'adipisci', 'adipisci']
```

4.2 Sorting a dictionary by value

By default, if you use `sorted` function on a `dict`, it will use keys to sort it. To sort by values, you can use operator.itemgetter(1) Return a callable object that fetches item from its operand using the operand's `__getitem__(` method. It could be used to sort results.

```
In [8]: import operator
        fruits = [('apple', 3), ('banana', 2), ('pear', 5), ('orange', 1)]
        getcount = operator.itemgetter(1)
        dict(sorted(fruits, key=getcount))
```

```
Out[8]: {'orange': 1, 'banana': 2, 'apple': 3, 'pear': 5}
```

`sorted` function has also a `reverse` optional argument.

```
In [9]: dict(sorted(fruits, key=getcount, reverse=True))
```

```
Out[9]: {'pear': 5, 'apple': 3, 'banana': 2, 'orange': 1}
```

4.2.1 Exercise 4.3

Create a function `reduce` to reduce the list of words returned by `map_words` and return a dictionary containing all words as keys and number of occurrences as values.

```
wordcount('sample.txt')
{'tempora': 2, 'non': 1, 'quisquam': 1, 'amet': 1, 'sit': 1}
```

```
In [10]: def reduce(sorted_words):
             " Compute word occurences from sorted list of words"

             res = {}
             current_word = None
             for word in sorted_words:
                 if word == current_word:
                     res[word] += 1
                 else:
                     res[word] = 1
```

```
                current_word = word
        return dict(sorted(res.items(), key=lambda v:v[1], reverse=True))

    reduce(map_words("sample.txt"))
```

```
Out[10]: {'ipsum': 75604,
          'est': 75594,
          'adipisci': 75558,
          'amet': 75517,
          'neque': 75467,
          'dolorem': 75320,
          'dolore': 75317,
          'labore': 75312,
          'etincidunt': 75306,
          'numquam': 75298,
          'eius': 75273,
          'consectetur': 75186,
          'sed': 75184,
          'dolor': 75137,
          'modi': 75101,
          'magnam': 75089,
          'sit': 75054,
          'voluptatem': 75046,
          'ut': 75034,
          'quisquam': 75031,
          'tempora': 74918,
          'porro': 74906,
          'non': 74891,
          'velit': 74889,
          'quiquia': 74863,
          'quaerat': 74816,
          'aliquam': 74732}
```

- `reduce` function using python exception KeyError

```
In [11]: def reduce(sorted_words):
             " Compute word occurences from sorted list of words"

             res = {}
             for word in sorted_words:
                 try:
                     res[word] += 1
                 except KeyError:
                     res[word] = 1

             return dict(sorted(res.items(), key=lambda v:v[1], reverse=True))

         reduce(map_words("sample.txt"))
```

```
Out[11]: {'ipsum': 75604,
          'est': 75594,
          'adipisci': 75558,
          'amet': 75517,
          'neque': 75467,
          'dolorem': 75320,
          'dolore': 75317,
          'labore': 75312,
```

```
'etincidunt': 75306,
'numquam': 75298,
'eius': 75273,
'consectetur': 75186,
'sed': 75184,
'dolor': 75137,
'modi': 75101,
'magnam': 75089,
'sit': 75054,
'voluptatem': 75046,
'ut': 75034,
'quisquam': 75031,
'tempora': 74918,
'porro': 74906,
'non': 74891,
'velit': 74889,
'quiquia': 74863,
'quaerat': 74816,
'aliquam': 74732}
```

You probably notice that this simple function is not easy to implement. Python standard library provides some features that can help.

4.3 Container datatypes

`collection` module implements specialized container datatypes providing alternatives to Python's general purpose built-in containers, `dict`, `list`, `set`, and `tuple`.

- `defaultdict` : dict subclass that calls a factory function to supply missing values
- `Counter` : dict subclass for counting hashable objects

4.3.1 defaultdict

When you implement the `wordcount` function you probably had some problem to append key-value pair to your `dict`. If you try to change the value of a key that is not present in the dict, the key is not automatically created.

You can use a `try-except` flow but the `defaultdict` could be a solution. This container is a `dict` subclass that calls a factory function to supply missing values. For example, using list as the default_factory, it is easy to group a sequence of key-value pairs into a dictionary of lists:

```
In [12]: from collections import defaultdict
         s = [('yellow', 1), ('blue', 2), ('yellow', 3), ('blue', 4), ('red', 1)]
         d = defaultdict(list)
         for k, v in s:
             d[k].append(v)

         dict(d)

Out[12]: {'yellow': [1, 3], 'blue': [2, 4], 'red': [1]}
```

4.3.2 Exercise 4.4

- Modify the `reduce` function you wrote above by using a defaultdict with the most suitable factory.

```
In [13]: from collections import defaultdict

         def reduce(sorted_words):
             " Reduce version using defaultdict, we use factory `int`"
             res = defaultdict(int)
             for word in sorted_words:
                 res[word] += 1

             return dict(sorted(res.items(), key=lambda v:v[1], reverse=True))

         reduce(map_words("sample.txt"))
```

```
Out[13]: {'ipsum': 75604,
          'est': 75594,
          'adipisci': 75558,
          'amet': 75517,
          'neque': 75467,
          'dolorem': 75320,
          'dolore': 75317,
          'labore': 75312,
          'etincidunt': 75306,
          'numquam': 75298,
          'eius': 75273,
          'consectetur': 75186,
          'sed': 75184,
          'dolor': 75137,
          'modi': 75101,
          'magnam': 75089,
          'sit': 75054,
          'voluptatem': 75046,
          'ut': 75034,
          'quisquam': 75031,
          'tempora': 74918,
          'porro': 74906,
          'non': 74891,
          'velit': 74889,
          'quiquia': 74863,
          'quaerat': 74816,
          'aliquam': 74732}
```

4.3.3 Counter

A Counter is a dict subclass for counting hashable objects. It is an unordered collection where elements are stored as dictionary keys and their counts are stored as dictionary values. Counts are allowed to be any integer value including zero or negative counts.

Elements are counted from an iterable or initialized from another mapping (or counter):

```
In [14]: from collections import Counter

         violet = dict(r=23,g=13,b=23)
         print(violet)
         cnt = Counter(violet)  # or Counter(r=238, g=130, b=238)
         print(cnt['c'])
         print(cnt['r'])
```

```
{'r': 23, 'g': 13, 'b': 23}
0
```

```
23
```

```
In [15]: print(*cnt.elements())
```

```
r r r r r r r r r r r r r r r r r r r r r r r g g g g g g g g g g g g g b b b b b b b b b b b b b b b b
```

```
In [16]: cnt.most_common(2)
```

```
Out[16]: [('r', 23), ('b', 23)]
```

```
In [17]: cnt.values()
```

```
Out[17]: dict_values([23, 13, 23])
```

4.3.4 Exercise 4.5

Use a `Counter` object to count words occurences in the sample text file.

```
In [18]: from collections import Counter

         def wordcounter(filename):

             " Wordcount function using the Counter type from collections"

             with open(filename) as f:
                 data = f.read()

             c = Counter(data.lower().replace("."," ").split())
             return dict(c.most_common())

         wordcounter("sample.txt")
```

```
Out[18]: {'ipsum': 75604,
          'est': 75594,
          'adipisci': 75558,
          'amet': 75517,
          'neque': 75467,
          'dolorem': 75320,
          'dolore': 75317,
          'labore': 75312,
          'etincidunt': 75306,
          'numquam': 75298,
          'eius': 75273,
          'consectetur': 75186,
          'sed': 75184,
          'dolor': 75137,
          'modi': 75101,
          'magnam': 75089,
          'sit': 75054,
          'voluptatem': 75046,
          'ut': 75034,
          'quisquam': 75031,
          'tempora': 74918,
          'porro': 74906,
          'non': 74891,
```

```
 'velit': 74889,
 'quiquia': 74863,
 'quaerat': 74816,
 'aliquam': 74732}
```

The Counter class is similar to bags or multisets in some Python libraries or other languages. We will see later how to use Counter-like objects in a parallel context.

4.4 Process multiple files

- Create several files containing `lorem` text named 'sample01.txt', 'sample02.txt'...
- If you process these files you return multiple dictionaries.
- You have to loop over them to sum occurences and return the resulted dict. To iterate on specific mappings, Python standard library provides some useful features in `itertools` module.
- itertools.chain(*mapped_values) could be used for treating consecutive sequences as a single sequence.

```
In [19]: import itertools, operator
         fruits = [('apple', 3), ('banana', 2), ('pear', 5), ('orange', 1)]
         vegetables = [('endive', 2), ('spinach', 1), ('celery', 5), ('carrot', 4)]
         getcount = operator.itemgetter(1)
         dict(sorted(itertools.chain(fruits,vegetables), key=getcount))
```

```
Out[19]: {'orange': 1,
          'spinach': 1,
          'banana': 2,
          'endive': 2,
          'apple': 3,
          'carrot': 4,
          'pear': 5,
          'celery': 5}
```

4.4.1 Exercise 4.6

- Write the program that creates files, processes and use `itertools.chain` to get the merged word count dictionary.

```
In [20]: import lorem

         for i in range(4): # write 4 sample text files
             with open(f"sample{i:02d}.txt", "w") as f:
                 f.write(lorem.text())
```

```
In [21]: from glob import glob

         samples = glob("*.txt")
```

```
In [22]: from itertools import chain

         words1 = map_words("sample01.txt")
         words2 = map_words("sample02.txt")

         reduce(chain(words1,words2)) # word count on two files
```

```
Out[22]: {'amet': 23,
          'magnam': 23,
          'voluptatem': 22,
          'porro': 21,
```

```
 'adipisci': 20,
 'numquam': 20,
 'eius': 19,
 'quisquam': 19,
 'sed': 19,
 'ipsum': 18,
 'quiquia': 18,
 'ut': 18,
 'dolorem': 17,
 'labore': 17,
 'est': 15,
 'non': 15,
 'quaerat': 15,
 'aliquam': 13,
 'dolor': 13,
 'etincidunt': 13,
 'modi': 13,
 'tempora': 13,
 'velit': 13,
 'neque': 12,
 'consectetur': 11,
 'sit': 11,
 'dolore': 10}
```

- wordcount on a list of files

```
In [23]: from itertools import chain
         from glob import glob

         reduce(chain(*[map_words(file) for file in glob("sample0*.txt")]))
```

```
Out[23]: {'voluptatem': 99,
          'porro': 97,
          'sed': 92,
          'quiquia': 90,
          'quisquam': 88,
          'velit': 88,
          'adipisci': 87,
          'dolorem': 86,
          'eius': 86,
          'modi': 86,
          'numquam': 86,
          'est': 85,
          'tempora': 85,
          'neque': 84,
          'etincidunt': 83,
          'dolor': 82,
          'non': 82,
          'labore': 80,
          'ipsum': 79,
          'quaerat': 79,
          'ut': 79,
          'consectetur': 74,
          'amet': 73,
          'sit': 73,
```

```
 'magnam': 69,
 'aliquam': 66,
 'dolore': 63}
```

4.4.2 Exercise 4.7

- Create the `wordcount` function in order to accept several files as arguments and return the result dict.

```
wordcount(file1, file2, file3, ...)
```

Hint: arbitrary argument lists

- Example of use of arbitrary argument list and arbitrary named arguments.

```
In [24]: def func( *args, **kwargs):
             for arg in args:
                 print(arg)

             print(kwargs)

         func( "3", [1,2], "bonjour", x = 4, y = "y")
```

```
3
[1, 2]
bonjour
{'x': 4, 'y': 'y'}
```

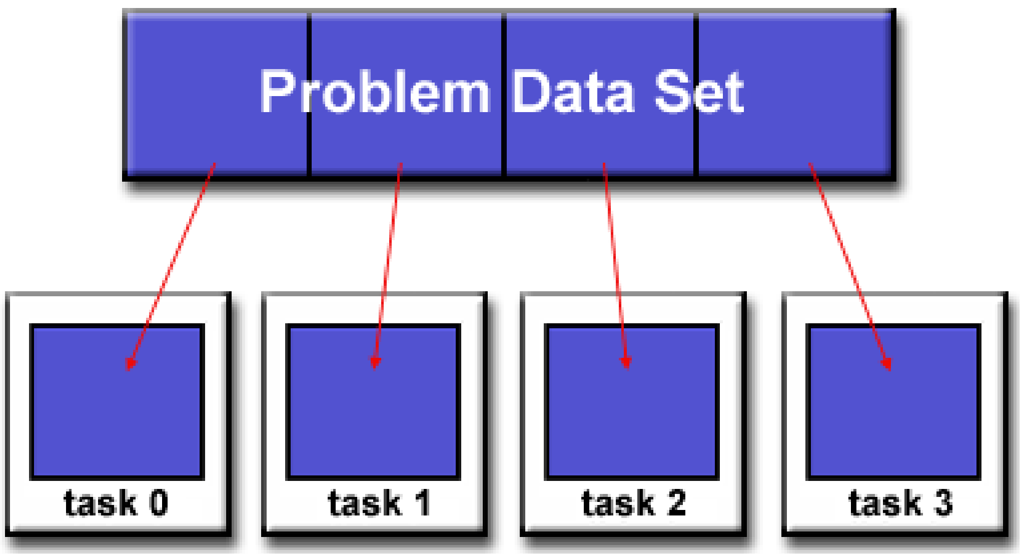
```
In [25]: from itertools import chain
         from glob import glob

         def wordcount(*args): # arbitrary argument list

             # MAP
             mapped_values = []
             for filename in args:
                 with open(filename) as f:
                     data = f.read()
                 words = data.lower().replace('.','').strip().split()
                 mapped_values.append(sorted(words))

             # REDUCE
             return reduce(chain(*mapped_values))

         wordcount(*glob("sample0*.txt"))
```

```
Out[25]: {'voluptatem': 99,
          'porro': 97,
          'sed': 92,
          'quiquia': 90,
          'quisquam': 88,
          'velit': 88,
          'adipisci': 87,
          'dolorem': 86,
          'eius': 86,
          'modi': 86,
          'numquam': 86,
          'est': 85,
```

```
 'tempora': 85,
 'neque': 84,
 'etincidunt': 83,
 'dolor': 82,
 'non': 82,
 'labore': 80,
 'ipsum': 79,
 'quaerat': 79,
 'ut': 79,
 'consectetur': 74,
 'amet': 73,
 'sit': 73,
 'magnam': 69,
 'aliquam': 66,
 'dolore': 63}
```

Chapter 5

Map Reduce



credits: https://computing.llnl.gov/tutorials/parallel_comp

5.1 map function example

The `map(func, seq)` Python function applies the function func to all the elements of the sequence seq. It returns a new list with the elements changed by func

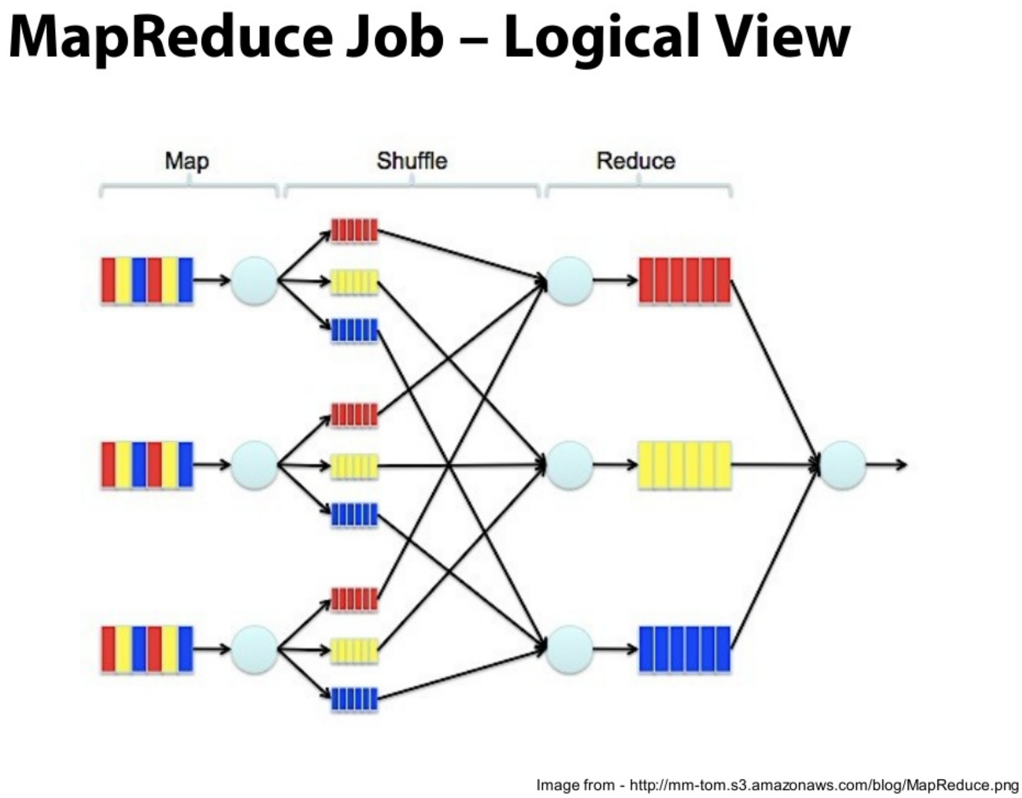
```
In [1]: def f(x):
            return x * x

        rdd = [2, 6, -3, 7]
        res = map(f, rdd )
        res  # Res is an iterator
```

```
Out[1]: <map at 0x7fdc381e2280>
```

```
In [2]: print(*res)
```

```
4 36 9 49
```

```
In [3]: from operator import mul
        rdd1, rdd2 = [2, 6, -3, 7], [1, -4, 5, 3]
        res = map(mul, rdd1, rdd2 ) # element wise sum of rdd1 and rdd2

In [4]: print(*res)

2 -24 -15 21
```



5.2 `functools.reduce` example

The function `reduce(func, seq)` continually applies the function func() to the sequence seq and return a single value. For example, reduce(f, [1, 2, 3, 4, 5]) calculates f(f(f(f(1,2),3),4),5).

```
In [5]: from functools import reduce
        from operator import add

        reduce(mul, rdd) # computes ((((1+2)+3)+4)+5)

Out[5]: -252

In [6]: p = 1
        for x in rdd:
            p *= x
        p

Out[6]: -252
```

```
In [7]: def g(x,y):
            return x * y

        reduce(g, rdd)
```

```
Out[7]: -252
```

```
In [8]: p = 1
        for x in rdd:
            p *= x
        p
```

```
Out[8]: -252
```

5.3 Weighted mean and Variance

If the generator of random variable X is discrete with probability mass function $x_1 \mapsto p_1, x_2 \mapsto p_2, \ldots, x_n \mapsto p_n$ then

$$\mathrm{Var}(X) = \left(\sum_{i=1}^{n} p_i x_i^2\right) - \mu^2,$$

where μ is the average value, i.e.

$$\mu = \sum_{i=1}^{n} p_i x_i.$$

```
In [9]: X = [5, 1, 2, 3, 1, 2, 5, 4]
        P = [0.05, 0.05, 0.15, 0.05, 0.15, 0.2, 0.1, 0.25]
```

Example of `zip`

```
In [10]: for x, p in zip(X, P):
             print(f" x = {x} ..... p = {p}")
```

```
 x = 5 … p = 0.05
 x = 1 … p = 0.05
 x = 2 … p = 0.15
 x = 3 … p = 0.05
 x = 1 … p = 0.15
 x = 2 … p = 0.2
 x = 5 … p = 0.1
 x = 4 … p = 0.25
```

```
In [11]: from itertools import zip_longest

         for x, p in zip_longest(X, [0.1], fillvalue=0.0):
             print(f" x = {x} ..... p = {p}")
```

```
 x = 5 … p = 0.1
 x = 1 … p = 0.0
 x = 2 … p = 0.0
 x = 3 … p = 0.0
 x = 1 … p = 0.0
 x = 2 … p = 0.0
 x = 5 … p = 0.0
 x = 4 … p = 0.0
```

```
In [12]: sum(P)

Out[12]: 0.9999999999999999
```

5.3.1 Exercise 5.1

- Write functions to compute the average value and variance using for loops

```
In [13]: def weighted_mean( X, P):
             return sum([x*p for x, p in zip(X,P)])

         weighted_mean(X,P)

Out[13]: 2.8

In [14]: def variance(X, P):
             mu = weighted_mean(X,P)
             return sum([p*x*x for x,p in zip(X,P)]) - mu**2

         variance(X, P)

Out[14]: 1.9600000000000017
```

5.3.2 Exercise 5.2

- Write functions to compute the average value and variance using `map` and `reduce`

```
In [15]: from operator import add, mul
         from functools import reduce

         def weighted_mean( X, P):
             return reduce(add,map(mul, X, P))

         weighted_mean(X,P)

Out[15]: 2.8

In [16]: def variance(X, P):
             mu = weighted_mean(X,P)
             return reduce(add,map(lambda x,p:p*x*x, X, P)) - mu**2

         variance(X, P)

Out[16]: 1.9600000000000017
```

5.3.3 Examples with filter

```
In [17]: res = filter( lambda p: p > 0.1, P)  # select p > 0.1
         print(*res)

0.15 0.15 0.2 0.25


In [18]: res = filter( lambda x: x % 3 == 0, range(15)) # select integer that can be divided by 3
         print(*res)

0 3 6 9 12
```

NB: Exercises above are just made to help to understand map-reduce process. This is a bad way to code a variance in Python. You should use Numpy instead.

5.4 Wordcount

We will modify the `wordcount` application into a map-reduce process.

The `map` process takes text files as input and breaks it into words. The `reduce` process sums the counts for each word and emits a single key/value with the word and sum.

We need to split the wordcount function we wrote in notebook 04 in order to use map and reduce.

In the following exercices we will implement in Python the Java example described in Hadoop documentation.

5.5 Map - Read file and return a key/value pairs

5.5.1 Exercise 5.3

Write a function `mapper` with a single file name as input that returns a sorted sequence of tuples (word, 1) values.

```
mapper('sample.txt')
[('adipisci', 1), ('adipisci', 1), ('adipisci', 1), ('adipisci', 1), ('adipisci', 1), ('adipisci', 1),
```

In [19]:
```python
import lorem
with open('sample.txt','w') as f:
    f.write(lorem.text())
```

In [20]:
```python
def mapper(filename):
    with open(filename) as f:
        data = f.read()

    data = data.strip().replace(".","").lower().split()

    return sorted([(w,1) for w in data])

mapper("sample.txt")
```

Out[20]:
```
[('adipisci', 1),
 ('adipisci', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('amet', 1),
 ('amet', 1),
 ('amet', 1),
 ('amet', 1),
 ('amet', 1),
 ('amet', 1),
 ('amet', 1),
 ('amet', 1),
 ('amet', 1),
```

```
('consectetur', 1),
('consectetur', 1),
('consectetur', 1),
('consectetur', 1),
('consectetur', 1),
('consectetur', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolore', 1),
('dolore', 1),
('dolore', 1),
('dolore', 1),
('dolore', 1),
('dolore', 1),
('dolore', 1),
('dolore', 1),
('dolore', 1),
('dolorem', 1),
('dolorem', 1),
('dolorem', 1),
('dolorem', 1),
('dolorem', 1),
('dolorem', 1),
('dolorem', 1),
('dolorem', 1),
('dolorem', 1),
('eius', 1),
('eius', 1),
('eius', 1),
('eius', 1),
('eius', 1),
('eius', 1),
('eius', 1),
('eius', 1),
('est', 1),
('est', 1),
('est', 1),
('est', 1),
('est', 1),
('est', 1),
('est', 1),
('etincidunt', 1),
('etincidunt', 1),
('etincidunt', 1),
('etincidunt', 1),
('etincidunt', 1),
('etincidunt', 1),
('etincidunt', 1),
```

```
('ipsum', 1),
('ipsum', 1),
('ipsum', 1),
('labore', 1),
('labore', 1),
('labore', 1),
('labore', 1),
('labore', 1),
('labore', 1),
('labore', 1),
('magnam', 1),
('magnam', 1),
('modi', 1),
('modi', 1),
('modi', 1),
('modi', 1),
('modi', 1),
('modi', 1),
('modi', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('non', 1),
('non', 1),
('non', 1),
('non', 1),
('non', 1),
('numquam', 1),
('numquam', 1),
('numquam', 1),
('numquam', 1),
('numquam', 1),
('numquam', 1),
('numquam', 1),
('numquam', 1),
('numquam', 1),
('numquam', 1),
('porro', 1),
('porro', 1),
('porro', 1),
('porro', 1),
('porro', 1),
('porro', 1),
('quaerat', 1),
('quaerat', 1),
```

```
('quaerat', 1),
('quaerat', 1),
('quiquia', 1),
('quiquia', 1),
('quiquia', 1),
('quiquia', 1),
('quiquia', 1),
('quiquia', 1),
('quiquia', 1),
('quisquam', 1),
('quisquam', 1),
('sed', 1),
('sed', 1),
('sed', 1),
('sed', 1),
('sed', 1),
('sed', 1),
('sed', 1),
('sed', 1),
('sed', 1),
('sit', 1),
('sit', 1),
('sit', 1),
('sit', 1),
('sit', 1),
('tempora', 1),
('tempora', 1),
('tempora', 1),
('tempora', 1),
('tempora', 1),
('tempora', 1),
('tempora', 1),
('tempora', 1),
('tempora', 1),
('tempora', 1),
('tempora', 1),
('ut', 1),
('ut', 1),
('ut', 1),
('ut', 1),
('ut', 1),
('ut', 1),
('ut', 1),
('ut', 1),
('velit', 1),
('velit', 1),
('velit', 1),
('velit', 1),
('velit', 1),
('velit', 1),
('velit', 1),
('voluptatem', 1),
('voluptatem', 1),
('voluptatem', 1),
```

```
('voluptatem', 1),
('voluptatem', 1)]
```

5.6 Partition

5.6.1 Exercise 5.4

Create a function named `partitioner` that stores the key/value pairs from `mapper` that group (word, 1) pairs into a list as:

```
partitioner(mapper('sample.txt'))
[('adipisci', [1, 1, 1, 1, 1, 1, 1]), ('aliquam', [1, 1, 1, 1, 1, 1, 1]), ('amet', [1, 1, 1, 1],...]
```

```
In [21]: from collections import defaultdict


         def partitioner(mapped_values):

             res = defaultdict(list)
             for w, c in mapped_values:
                 res[w].append(c)

             return res.items()


         partitioner(mapper('sample.txt'))
```

```
Out[21]: dict_items([('adipisci', [1, 1]), ('aliquam', [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]), ('amet',
```

5.7 Reduce - Sums the counts and returns a single key/value (word, sum).

5.7.1 Exercice 5.5

Write the function `reducer` that read a tuple (word,[1,1,1,..,1]) and sum the occurrences of word to a final count, and then output the tuple (word,occurences).

```
reducer(('hello',[1,1,1,1,1])
('hello',5)
```

```
In [22]: from operator import itemgetter


         def reducer( item ):
             w, v = item
             return (w,len(v))

         reducer(('hello',[1,1,1,1,1]))
```

```
Out[22]: ('hello', 5)
```

5.8 Process several files

Let's create 8 files `sample[0-7].txt`. Set most common words at the top of the output list.

```
In [23]: from lorem import text
         for i in range(8):
             with open("sample{0:02d}.txt".format(i), "w") as f:
                 f.write(text())
```

```
In [24]: import glob
         files = sorted(glob.glob('sample0*.txt'))
         files
```

```
Out[24]: ['sample00.txt',
          'sample01.txt',
          'sample02.txt',
          'sample03.txt',
          'sample04.txt',
          'sample05.txt',
          'sample06.txt',
          'sample07.txt',
          'sample08.txt',
          'sample09.txt']
```

5.8.1 Exercise 5.6

- Use functions implemented above to count (word, occurences) by using a for loops over files and partitioned data.

```
In [25]: from itertools import chain

         def wordcount(files):

             mapped_values = [mapper(file) for file in files]

             partioned_values = partitioner(chain(*mapped_values))

             return sorted([reducer(val) for val in partioned_values],
                           key = itemgetter(1),
                           reverse = True)

         wordcount(files)
```

```
Out[25]: [('non', 124),
          ('quaerat', 118),
          ('dolore', 115),
          ('neque', 111),
          ('porro', 108),
          ('quisquam', 108),
          ('adipisci', 107),
          ('ipsum', 104),
          ('numquam', 104),
          ('aliquam', 102),
          ('modi', 102),
          ('velit', 102),
          ('dolorem', 101),
          ('voluptatem', 101),
          ('est', 100),
          ('amet', 99),
          ('eius', 98),
          ('quiquia', 98),
```

```
('consectetur', 97),
('etincidunt', 96),
('tempora', 95),
('ut', 94),
('sed', 89),
('sit', 87),
('dolor', 85),
('magnam', 85),
('labore', 80)]
```

5.8.2 Exercise 5.7

- This time use `map` function to apply mapper and reducer.

```
In [26]: def wordcount(files):

             mapped_values = map(mapper, files)
             partioned_values = partitioner(chain(*mapped_values))

             return sorted( map(reducer, partioned_values),
                            key=itemgetter(1),
                            reverse=True)


         wordcount(files)

Out[26]: [('non', 124),
          ('quaerat', 118),
          ('dolore', 115),
          ('neque', 111),
          ('porro', 108),
          ('quisquam', 108),
          ('adipisci', 107),
          ('ipsum', 104),
          ('numquam', 104),
          ('aliquam', 102),
          ('modi', 102),
          ('velit', 102),
          ('dolorem', 101),
          ('voluptatem', 101),
          ('est', 100),
          ('amet', 99),
          ('eius', 98),
          ('quiquia', 98),
          ('consectetur', 97),
          ('etincidunt', 96),
          ('tempora', 95),
          ('ut', 94),
          ('sed', 89),
          ('sit', 87),
          ('dolor', 85),
          ('magnam', 85),
          ('labore', 80)]
```

Chapter 6

Parallel Computation

6.1 Parallel computers

- Multiprocessor/multicore: several processors work on data stored in shared memory
- Cluster: several processor/memory units work together by exchanging data over a network
- Co-processor: a general-purpose processor delegates specific tasks to a special-purpose processor (GPU)

6.2 Parallel Programming

- Decomposition of the complete task into independent subtasks and the data flow between them.
- Distribution of the subtasks over the processors minimizing the total execution time.
- For clusters: distribution of the data over the nodes minimizing the communication time.
- For multiprocessors: optimization of the memory access patterns minimizing waiting times.
- Synchronization of the individual processes.

6.3 MapReduce

```
In [1]: from time import sleep
        def f(x):
            sleep(1)
            return x*x
        L = list(range(8))
        L

Out[1]: [0, 1, 2, 3, 4, 5, 6, 7]
```

```
In [2]: %time sum(f(x) for x in L)

CPU times: user 2.5 ms, sys: 326 µs, total: 2.82 ms
Wall time: 8.01 s


Out[2]: 140
```

```
In [3]: %time sum(map(f,L))

CPU times: user 3.21 ms, sys: 0 ns, total: 3.21 ms
Wall time: 8.01 s


Out[3]: 140
```

6.4 Multiprocessing

`multiprocessing` is a package that supports spawning processes.

We can use it to display how many concurrent processes you can launch on your computer.

```
In [4]: from multiprocessing import cpu_count

        cpu_count()
```

```
Out[4]: 2
```

6.5 Futures

The `concurrent.futures` module provides a high-level interface for asynchronously executing callables.

The asynchronous execution can be performed with: - **threads**, using ThreadPoolExecutor, - separate **processes**, using ProcessPoolExecutor. Both implement the same interface, which is defined by the abstract Executor class.

`concurrent.futures` can't launch **processes** on windows. Windows users must install loky.

```
In [5]: %%file pmap.py
        from concurrent.futures import ProcessPoolExecutor
        from time import sleep, time

        def f(x):
            sleep(1)
            return x*x

        L = list(range(8))

        if __name__ == '__main__':

            begin = time()
            with ProcessPoolExecutor() as pool:

                result = sum(pool.map(f, L))
            end = time()

            print(f"result = {result} and time = {end-begin}")
```

```
Overwriting pmap.py
```

```
In [6]: import sys
        !{sys.executable} pmap.py
```

```
result = 140 and time = 4.0100157260894775
```

- `ProcessPoolExecutor` launches one slave process per physical core on the computer.
- `pool.map` divides the input list into chunks and puts the tasks (function + chunk) on a queue.
- Each slave process takes a task (function + a chunk of data), runs map(function, chunk), and puts the result on a result list.
- `pool.map` on the master process waits until all tasks are handled and returns the concatenation of the result lists.

```
In [7]: %%time
        from concurrent.futures import ThreadPoolExecutor
```

```
with ThreadPoolExecutor() as pool:

    results = sum(pool.map(f, L))

print(results)
```

```
140
CPU times: user 0 ns, sys: 6.98 ms, total: 6.98 ms
Wall time: 2 s
```

6.6 Thread and Process: Differences

- A **process** is an instance of a running program.
- **Process** may contain one or more **threads**, but a **thread** cannot contain a **process**.
- **Process** has a self-contained execution environment. It has its own memory space.
- Application running on your computer may be a set of cooperating **processes**.
- **Process** don't share its memory, communication between **processes** implies data serialization.
- A **thread** is made of and exist within a **process**; every **process** has at least one **thread**.
- Multiple **threads** in a **process** share resources, which helps in efficient communication between **threads**.
- **Threads** can be concurrent on a multi-core system, with every core executing the separate **threads** simultaneously.

6.7 The Global Interpreter Lock (GIL)

- The Python interpreter is not thread safe.
- A few critical internal data structures may only be accessed by one thread at a time. Access to them is protected by the GIL.
- Attempts at removing the GIL from Python have failed until now. The main difficulty is maintaining the C API for extension modules.
- Multiprocessing avoids the GIL by having separate processes which each have an independent copy of the interpreter data structures.
- The price to pay: serialization of tasks, arguments, and results.

6.8 Weighted mean and Variance

6.8.1 Exercise 6.1

Use `ThreadPoolExecutor` to parallelized functions written in notebook 05

```
In [8]: X = [5, 1, 2, 3, 1, 2, 5, 4]
        P = [0.05, 0.05, 0.15, 0.05, 0.15, 0.2, 0.1, 0.25]
```

```
In [9]: from operator import add, mul
        from functools import reduce
        from concurrent.futures import ThreadPoolExecutor as pool

        def weighted_mean( X, P):
```

```
            with pool() as p:
                w1 = p.map(mul, X, P)

            return reduce(add,w1)

        weighted_mean(X,P)
```

Out[9]: 2.8

```
In [10]: def variance(X, P):
             mu = weighted_mean(X,P)
             with pool() as p:
                 w2 = p.map(lambda x,p:p*x*x, X, P)
             return reduce(add,w2) - mu**2

         variance(X, P)
```

Out[10]: 1.9600000000000017

```
In [11]: import numpy as np
         x = np.array(X)
         p = np.array(P)
         np.average( x, weights=p)
```

Out[11]: 2.8

```
In [12]: var =np.sum(p*x**2) - np.average( x, weights=p)**2
         var
```

Out[12]: 1.9600000000000017

6.9 Wordcount

```
In [13]: from glob import glob
         from collections import defaultdict
         from operator import itemgetter
         from itertools import chain
         from concurrent.futures import ThreadPoolExecutor


         def mapper(filename):
             " split text to list of key/value pairs (word,1)"
             with open(filename) as f:
                 data = f.read()

             data = data.strip().replace(".","").lower().split()

             return sorted([(w,1) for w in data])


         def partitioner(mapped_values):
             """ get lists from mapper and create a dict with
             (word,[1,1,1])"""

             res = defaultdict(list)
             for w, c in mapped_values:
                 res[w].append(c)

             return res.items()
```

```
def reducer( item ):
    """ Compute words occurences from dict computed
    by partioner
    """
    w, v = item
    return (w,len(v))
```

6.10 Parallel map

- Let's improve the `mapper` function by print out inside the function the current process name.

Example

```
In [14]: import multiprocessing as mp
         from concurrent.futures import ThreadPoolExecutor
         #from loky import ProcessPoolExecutor
         def process_name(n):
             " prints out the current process name "
             print(mp.current_process().name)

         with ProcessPoolExecutor() as e:
             _ = e.map(process_name, range(mp.cpu_count()))
```

```
        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-14-2096cf4534b5> in <module>
          6     print(mp.current_process().name)
          7
    ----> 8 with ProcessPoolExecutor() as e:
          9     _ = e.map(process_name, range(mp.cpu_count()))


        NameError: name 'ProcessPoolExecutor' is not defined
```

6.10.1 Exercise 6.2

- Modify the mapper function by adding this print.

```
In [15]: def mapper(filename):
             " split text to list of key/value pairs (word,1)"

             print(f"{mp.current_process().name} : {filename}")
             with open(filename) as f:
                 data = f.read()

             data = data.strip().replace(".","").lower().split()

             return sorted([(w,1) for w in data])
```

6.11 Parallel reduce

- For parallel reduce operation, data must be aligned in a container. We already created a `partitioner` function that returns this container.

6.11.1 Exercise 6.3

Write a parallel program that uses the three functions above using `ProcessPoolExecutor`. It reads all the "sample*.txt" files. Map and reduce steps are parallel.

```
In [16]: from concurrent.futures import ThreadPoolExecutor

         def wordcount(files):

             with ThreadPoolExecutor() as e:

                 mapped_values = e.map(mapper, files)
                 partioned_values = partitioner(chain(*mapped_values))
                 occurences = e.map(reducer, partioned_values)

             return sorted(occurences,
                           key=itemgetter(1),
                           reverse=True)

         files = glob("sample*.txt")
         wordcount(files)

Out[16]: []
```

6.12 Increase volume of data

Due to the proxy, code above is not runnable on workstations

6.12.1 Getting the data

- The Latin Library contains a huge collection of freely accessible Latin texts. We get links on the Latin Library's homepage ignoring some links that are not associated with a particular author.

```
In [17]: from bs4 import BeautifulSoup  # web scraping library
         from urllib.request import *

         base_url = "http://www.thelatinlibrary.com/"
         home_content = urlopen(base_url)

         soup = BeautifulSoup(home_content, "lxml")
         author_page_links = soup.find_all("a")
         author_pages = [ap["href"] for i, ap in enumerate(author_page_links) if i < 49]
```

6.12.2 Generate html links

- Create a list of all links pointing to Latin texts. The Latin Library uses a special format which makes it easy to find the corresponding links: All of these links contain the name of the text author.

```
In [18]: ap_content = list()
         for ap in author_pages:
             ap_content.append(urlopen(base_url + ap))
```

```
book_links = list()
for path, content in zip(author_pages, ap_content):
    author_name = path.split(".")[0]
    ap_soup = BeautifulSoup(content, "lxml")
    book_links += ([link for link in ap_soup.find_all("a", {"href": True}) if author_name in link["href"]]
```

6.12.3 Download webpages content

```
In [19]: from urllib.error import HTTPError

         num_pages = 100

         for i, bl in enumerate(book_links[:num_pages]):
             print("Getting content " + str(i + 1) + " of " + str(num_pages), end="\r", flush=True)
             try:
                 content = urlopen(base_url + bl["href"]).read()
                 with open(f"book-{i:03d}.dat","wb") as f:
                     f.write(content)
             except HTTPError as err:
                 print("Unable to retrieve " + bl["href"] + ".")
                 continue
```

```
Getting content 100 of 100
```

6.12.4 Extract data files

- I already put the content of pages in files named book-*.txt
- You can extract data from the archive by running the cell below

```
import os  # library to get directory and file paths
import tarfile # this module makes possible to read and write tar archives

def extract_data():
    datadir = os.path.join('data','latinbooks')
    if not os.path.exists(datadir):
       print("Extracting data...")
       tar_path = os.path.join('data', 'latinbooks.tgz')
       with tarfile.open(tar_path, mode='r:gz') as books:
          books.extractall('data')

extract_data() # this function call will extract text files in data/latinbooks
```

6.12.5 Read data files

```
In [20]: from glob import glob
         files = glob('book*.dat')
         texts = list()
         for file in files:
             with open(file,'rb') as f:
                 text = f.read()
             texts.append(text)
```

6.12.6 Extract the text from html and split the text at periods to convert it into sentences.

```
In [21]: %%time
         from bs4 import BeautifulSoup
```

```
sentences = list()

for i, text in enumerate(texts):
    print("Document " + str(i + 1) + " of " + str(len(texts)), end="\r", flush=True)
    textSoup = BeautifulSoup(text, "lxml")
    paragraphs = textSoup.find_all("p", attrs={"class":None})
    prepared = ("".join([p.text.strip().lower() for p in paragraphs[1:-1]]))
    for t in prepared.split("."):
        part = "".join([c for c in t if c.isalpha() or c.isspace()])
        sentences.append(part.strip())

# print first and last sentence to check the results
print(sentences[0])
print(sentences[-1])
```

```
sed nimirum nihil fortuna rennuente licet homini natu dexterum provenire nec consilio prudenti vel remed

CPU times: user 1.73 s, sys: 53.1 ms, total: 1.78 s
Wall time: 1.72 s
```

6.12.7 Exercise 6.4

Parallelize this last process using `concurrent.futures`.

```
In [22]: %%time
         from bs4 import BeautifulSoup
         from concurrent.futures import ThreadPoolExecutor as pool

         def sentence_mapper(text):
             sentences = list()
             textSoup = BeautifulSoup(text, "lxml")
             paragraphs = textSoup.find_all("p", attrs={"class":None})
             prepared = ("".join([p.text.strip().lower() for p in paragraphs[1:-1]]))
             for t in prepared.split("."):
                 part = "".join([c for c in t if c.isalpha() or c.isspace()])
                 sentences.append(part.strip())
             return sentences

         # parallel map
         with pool(4) as p:

             mapped_sentences = p.map(sentence_mapper, texts)

         # reduce
         sentences = reduce(add, mapped_sentences )

         # print first and last sentence to check the results
         print(sentences[0])
         print(sentences[-1])
```

```
sed nimirum nihil fortuna rennuente licet homini natu dexterum provenire nec consilio prudenti vel remed

CPU times: user 2.46 s, sys: 1.07 s, total: 3.53 s
Wall time: 2.56 s
```

6.13 References

- Using Conditional Random Fields and Python for Latin word segmentation

Chapter 7

Asynchronous Processing

While many parallel applications can be described as maps, some can be more complex. In this section we look at the asynchronous `concurrent.futures` interface, which provides a simple API for ad-hoc parallelism. This is useful for when your computations don't fit a regular pattern.

7.0.1 Executor.submit

The `submit` method starts a computation in a separate thread or process and immediately gives us a `Future` object that refers to the result. At first, the future is pending. Once the function completes the future is finished.

We collect the result of the task with the `.result()` method, which does not return until the results are available.

```
In [1]: %%time
        from time import sleep

        def slowadd(a, b, delay=1):
            sleep(delay)
            return a + b

        slowadd(1,1)
```

```
CPU times: user 0 ns, sys: 1.48 ms, total: 1.48 ms
Wall time: 1 s
```

```
Out[1]: 2
```

```
In [2]: from concurrent.futures import ThreadPoolExecutor

        e = ThreadPoolExecutor()
        future = e.submit(slowadd, 1, 2)
        future
```

```
Out[2]: <Future at 0x7f59ad1de490 state=running>
```

```
In [3]: future.result()
```

```
Out[3]: 3
```

7.0.2 Submit many tasks, receive many futures

Because submit returns immediately we can submit many tasks all at once and they will execute in parallel.

```
In [4]: %%time
        results = [slowadd(i, i, delay=1) for i in range(8)]
        print(results)

[0, 2, 4, 6, 8, 10, 12, 14]
CPU times: user 0 ns, sys: 3.97 ms, total: 3.97 ms
Wall time: 8.01 s
```

```
In [5]: %%time
        e = ThreadPoolExecutor()
        futures = [e.submit(slowadd, i, i, delay=1) for i in range(8)]
        results = [f.result() for f in futures]
        print(results)

[0, 2, 4, 6, 8, 10, 12, 14]
CPU times: user 4.26 ms, sys: 0 ns, total: 4.26 ms
Wall time: 2 s
```

- Submit fires off a single function call in the background, returning a future.
- When we combine submit with a single for loop we recover the functionality of map.
- When we want to collect our results we replace each of our futures, `f`, with a call to `f.result()`
- We can combine submit with multiple for loops and other general programming to get something more general than map.

7.0.3 Exercise 7.1

Parallelize the following code with e.submit

1. Replace the `results` list with a list called `futures`
2. Replace calls to `slowadd` and `slowsub` with `e.submit` calls on those functions
3. At the end, block on the computation by recreating the `results` list by calling `.result()` on each future in the `futures` list.

```
In [6]: %%time
        from time import sleep

        def slowadd(a, b, delay=1):
            sleep(delay)
            return a + b

        def slowsub(a, b, delay=1):
            sleep(delay)
            return a - b

        results = []
        for i in range(4):
            for j in range(4):
                if i < j:
                    results.append(slowadd(i, j, delay=1))
                elif i > j:
                    results.append(slowsub(i, j, delay=1))

        print(results)
```

```
[1, 2, 3, 1, 3, 4, 2, 1, 5, 3, 2, 1]
CPU times: user 4.64 ms, sys: 124 µs, total: 4.76 ms
Wall time: 12 s
```

7.1 Extract daily stock data from google

```
In [7]: import os  # library to get directory and file paths
        import tarfile # this module makes possible to read and write tar archives

        def extract_data(name, where):
            datadir = os.path.join(where,name)
            if not os.path.exists(datadir):
               print("Extracting data...")
               tar_path = os.path.join(where, name+'.tgz')
               with tarfile.open(tar_path, mode='r:gz') as data:
                  data.extractall(where)

        extract_data('daily-stock','data') # this function call will extract json files
```

7.2 Convert data to pandas DataFrames and save it in hdf5 files

HDF5 is a data model, library, and file format for storing and managing data. This format is widely used and is supported by many languages and platforms.

```
In [8]: import json
        import pandas as pd
        import os, glob

        here = os.getcwd()
        datadir = os.path.join(here,'data','daily-stock')
        filenames = sorted(glob.glob(os.path.join(datadir, '*.json')))
        filenames
```

```
Out[8]: ['/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aet.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/afl.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aig.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/al.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/amgn.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/avy.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/b.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/bwa.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ge.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hal.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hp.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hpq.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ibm.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jbl.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jpm.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/luv.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/met.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/pcg.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/tgt.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/usb.json',
         '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/xom.json']
```

7.2.1 Sequential version

```
In [9]: %%time
        from concurrent.futures import ProcessPoolExecutor
        import json
        import pandas as pd

        def load_parse_store(fn):

            with open(fn) as f:
                data = [json.loads(line) for line in f] # load

            df = pd.DataFrame(data) # parse

            out_filename = fn[:-5] + '.h5'
            df.to_hdf(out_filename, '/data') # store
            print("Finished : %s" % (out_filename.split(os.path.sep)[-1]))

            return True


        results = [load_parse_store(file) for file in filenames]
```

```
Finished : aet.h5
Finished : afl.h5
Finished : aig.h5
Finished : al.h5
Finished : amgn.h5
Finished : avy.h5
Finished : b.h5
Finished : bwa.h5
Finished : ge.h5
Finished : hal.h5
Finished : hp.h5
Finished : hpq.h5
Finished : ibm.h5
Finished : jbl.h5
Finished : jpm.h5
Finished : luv.h5
Finished : met.h5
Finished : pcg.h5
Finished : tgt.h5
Finished : usb.h5
Finished : xom.h5
CPU times: user 8.25 s, sys: 1.05 s, total: 9.29 s
Wall time: 8.88 s
```

7.2.2 Exercise 7.2

Parallelize the loop above using `ThreadPoolExecutor` and `map`.

7.3 Read files and load dataframes.

```
In [10]: filenames = sorted(glob.glob(os.path.join('data', 'daily-stock', '*.h5')))
         series ={}
```

```
for fn in filenames:
    series[fn] = pd.read_hdf(fn)['close']
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-10-2dab4a15ac74> in <module>
      2 series ={}
      3 for fn in filenames:
----> 4     series[fn] = pd.read_hdf(fn)['close']


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/io/pytables.py in read_hd
    400             for group_to_check in groups[1:]:
    401                 if not _is_metadata_of(group_to_check, candidate_only_group):
--> 402                     raise ValueError(
    403                         "key must be provided when HDF5 "
    404                         "file contains multiple datasets."


ValueError: key must be provided when HDF5 file contains multiple datasets.
```

7.4 Application

Given our HDF5 files from the last section we want to find the two datasets with the greatest pair-wise correlation. This forces us to consider all $n \times (n-1)$ possibilities.

In [11]:
```
%%time

results = {}

for a in filenames:
    for b in filenames:
        if a != b:
            results[a, b] = series[a].corr(series[b])

((a, b), corr) = max(results.items(), key=lambda kv: kv[1])
print("%s matches with %s with correlation %f" % (a, b, corr))
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<timed exec> in <module>


KeyError: 'data/daily-stock/aet.h5'
```

We use matplotlib to visually inspect the highly correlated timeseries

```
In [12]: %matplotlib inline
         import matplotlib.pyplot as plt
         plt.figure(figsize=(10, 4))
         plt.plot(series[a]/series[a].max())
         plt.plot(series[b]/series[b].max())
         plt.xticks(visible=False);


         ---------------------------------------------------------------------------

         KeyError                                  Traceback (most recent call last)

         <ipython-input-12-1d4fb0c2882b> in <module>
           2 import matplotlib.pyplot as plt
           3 plt.figure(figsize=(10, 4))
     ----> 4 plt.plot(series[a]/series[a].max())
           5 plt.plot(series[b]/series[b].max())
           6 plt.xticks(visible=False);


         KeyError: 'data/daily-stock/aet.h5'


<Figure size 720x288 with 0 Axes>
```

7.5 Analysis

This computation starts out by loading data from disk. We already know how to parallelize it:

```
series = {}
for fn in filenames:
    series[fn] = pd.read_hdf(fn)['x']
```

It follows with a doubly nested for loop with an if statement.

```
results = {}
for a in filenames:
    for b in filenames:
        if a != b:
            results[a, b] = series[a].corr(series[b])
```

It *is* possible to solve this problem with `map`, but it requires some cleverness. Instead we'll learn `submit`, an interface to start individual function calls asynchronously.

It finishes with a reduction on small data. This part is fast enough.

```
((a, b), corr) = max(results.items(), key=lambda kv: kv[1])
```

```
In [13]: %%time

         from concurrent.futures import ThreadPoolExecutor as PoolExecutor


         e = PoolExecutor()

         def corr( serie_a, serie_b):
```

```
    return serie_a.corr(serie_b)

futures = {}

for a in filenames:
    for b in filenames:
        if a != b:
            futures[a, b] = e.submit( corr, series[a], series[b])

results = {k : f.result() for k, f in futures.items()}


((a, b), corr) = max(results.items(), key=lambda kv: kv[1])
print("%s matches with %s with correlation %f" % (a, b, corr))
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<timed exec> in <module>


KeyError: 'data/daily-stock/aet.h5'
```

7.5.1 Exercise 7.3

- Parallelize pair-wise correlations with `e.submit`
- Implement two versions one using Processes, another with Threads by replacing `e` with a ProcessPoolExecutor:

Threads

```
from concurrent.futures import ThreadPoolExecutor
e = ThreadPoolExecutor(4)
```

Processes

Be careful, a `ProcessPoolExecutor` does not run in the jupyter notebook cell. You must run your file in a terminal.

```
from concurrent.futures import ProcessPoolExecutor
e = ProcessPoolExecutor(4)
```

- How does performance vary?

7.6 Some conclusions about futures

- `submit` functions can help us to parallelize more complex applications
- It didn't actually speed up the code very much
- Threads and Processes give some performance differences
- This is not very robust.

Chapter 8

Dask

- process data that doesn't fit into memory by breaking it into blocks and specifying task chains
- parallelize execution of tasks across cores and even nodes of a cluster
- move computation to the data rather than the other way around, to minimize communication overheads

http://dask.pydata.org/en/latest/

```
In [1]: import dask
        import dask.multiprocessing
```

8.1 Define two slow functions

```
In [2]: from time import sleep

        def slowinc(x, delay=1):
            sleep(delay)
            return x + 1

        def slowadd(x, y, delay=1):
            sleep(delay)
            return x + y
```

```
In [3]: %%time
        x = slowinc(1)
        y = slowinc(2)
        z = slowadd(x, y)
```

```
CPU times: user 1.36 ms, sys: 275 µs, total: 1.64 ms
Wall time: 3 s
```

8.2 Parallelize with dask.delayed

- Functions wrapped by `dask.delayed` don't run immediately, but instead put those functions and arguments into a task graph.
- The result is computed separately by calling the `.compute()` method.

```
In [4]: from dask import delayed
```

```
In [5]: x = delayed(slowinc)(1)
        y = delayed(slowinc)(2)
        z = delayed(slowadd)(x, y)
```

In [6]:
```
%%time
z.compute()
```

```
CPU times: user 7.88 ms, sys: 0 ns, total: 7.88 ms
Wall time: 2.01 s
```

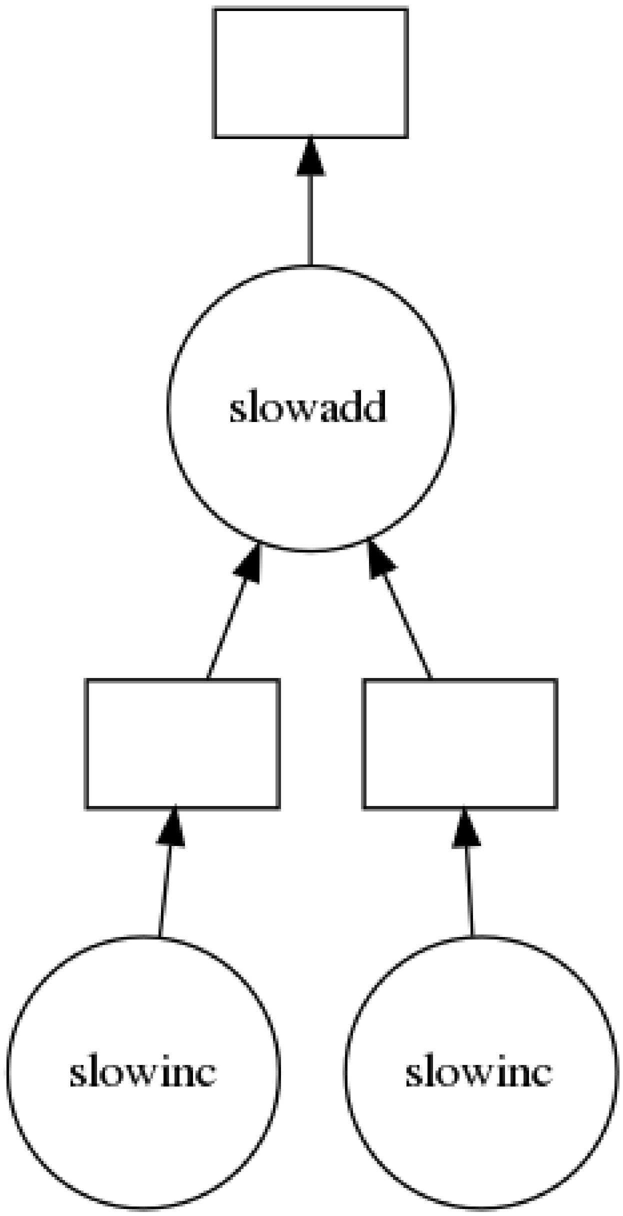
Out[6]:
```
5
```

8.3 Dask graph

- Contains description of the calculations necessary to produce the result.
- The z object is a lazy Delayed object. This object holds everything we need to compute the final result. We can compute the result with .compute() as above or we can visualize the task graph for this value with .visualize().

In [7]:
```
z.visualize()
```

Out[7]:



8.4 Parallelize a loop

In [8]:
```
%%time
data = list(range(8))

results = []

for x in data:
    y = slowinc(x)
    results.append(y)
```

```
        total = sum(results)
        total
```

```
CPU times: user 3.44 ms, sys: 0 ns, total: 3.44 ms
Wall time: 8.01 s
```

```
Out[8]: 36
```

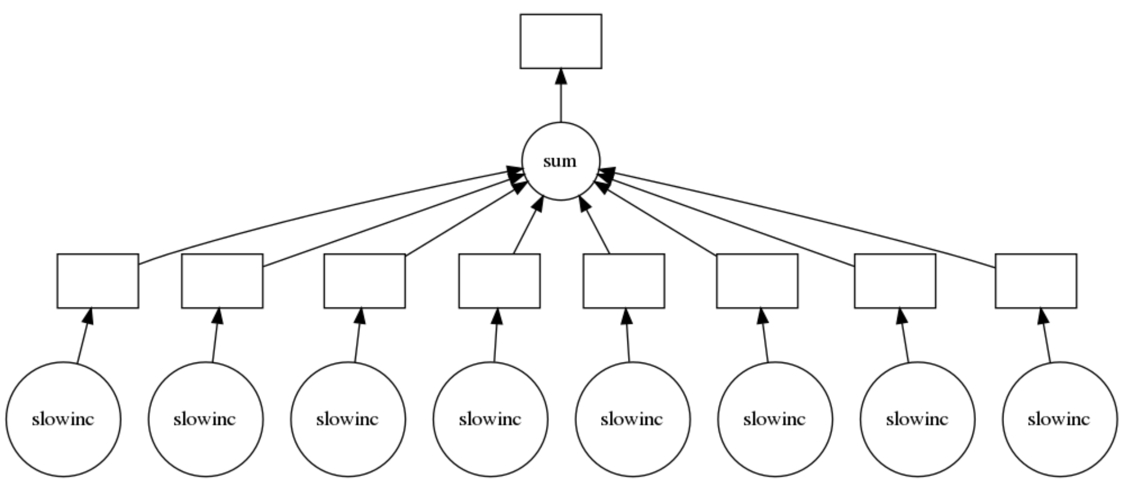
8.4.1 Exercise 8.1

- Parallelize this by appending the delayed `slowinc` calls to the list `results`.
- Display the graph of `total` computation
- Compute time elapsed for the computation.

```
In [9]: from dask import delayed

        futures = []

        for x in data:
            y = delayed(slowinc)(x)
            futures.append(y)

        total = delayed(sum)(futures)
```

```
In [10]: total.visualize()
```

```
Out[10]:
```




```
In [11]: %time total.compute()
```

```
CPU times: user 0 ns, sys: 6.9 ms, total: 6.9 ms
Wall time: 4.01 s
```

```
Out[11]: 36
```

8.5 Decorator

It is also common to see the delayed function used as a decorator. Same example:

```
In [12]: %%time

         @dask.delayed
         def slowinc(x, delay=1):
             sleep(delay)
             return x + 1

         @dask.delayed
         def slowadd(x, y, delay=1):
             sleep(delay)
             return x + y

         x = slowinc(1)
         y = slowinc(2)
         z = slowadd(x, y)
         z.compute()
```

```
CPU times: user 4 ms, sys: 288 µs, total: 4.29 ms
Wall time: 2 s
```

```
Out[12]: 5
```

8.6 Control flow

- Delay only some functions, running a few of them immediately. This is helpful when those functions are fast and help us to determine what other slower functions we should call.
- In the example below we iterate through a list of inputs. If that input is even then we want to call `half`. If the input is odd then we want to call `odd_process`. This iseven decision to call `half` or `odd_process` has to be made immediately (not lazily) in order for our graph-building Python code to proceed.

```
In [13]: from random import randint
         import dask.delayed

         @dask.delayed
         def half(x):
             sleep(1)
             return x // 2

         @dask.delayed
         def odd_process(x):
             sleep(1)
             return 3*x+1

         def is_even(x):
             return not x % 2

         data = [randint(0,100) for i in range(8)]
         data
```

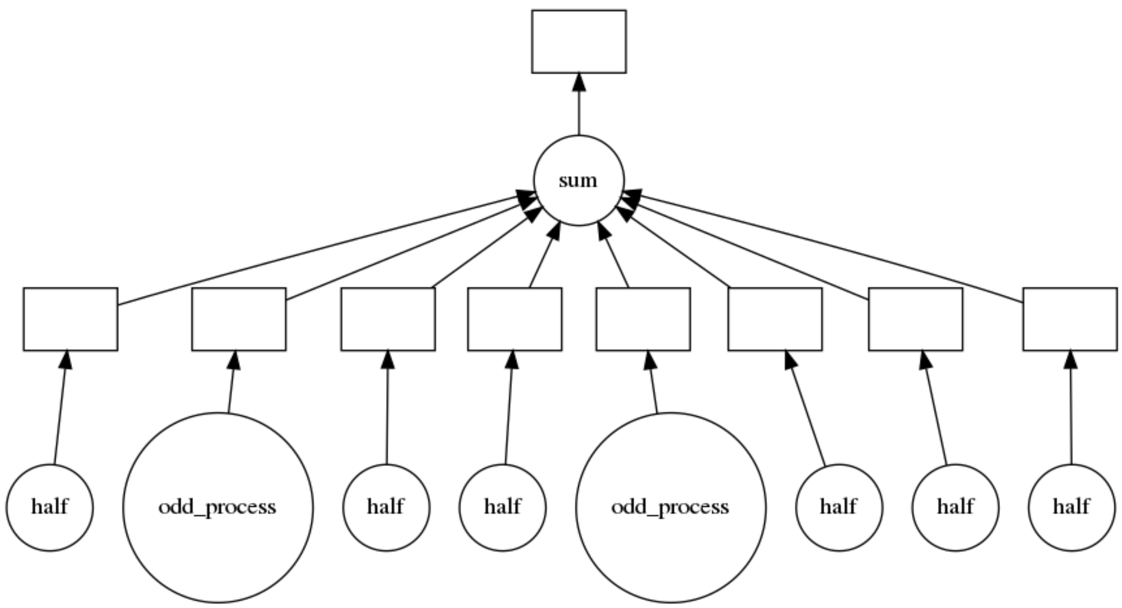
```
Out[13]: [86, 81, 77, 10, 88, 100, 52, 8]
```

8.6.1 Exercise 8.2

- Parallelize the sequential code above using dask.delayed
- You will need to delay some functions, but not all
- Visualize and check the computed result

```
In [14]: results = []
         for x in data:
             if is_even(x):
                 y = half(x)
             else:
                 y = odd_process(x)
             results.append(y)

         total = delayed(sum)(results)
         total.visualize()
```

Out[14]:



8.6.2 Exercise 8.3

- Parallelize the hdf5 conversion from json files
- Create a function `convert_to_hdf`
- Use dask.compute function on delayed calls of the funtion created list
- Is it really faster as expected ?

Hint: Read Delayed Best Practices

```
In [15]: import os  # library to get directory and file paths
         import tarfile # this module makes possible to read and write tar archives

         def extract_data(name, where):
             datadir = os.path.join(where,name)
             if not os.path.exists(datadir):
```

```
            print("Extracting data...")
            tar_path = os.path.join(where, name+'.tgz')
            with tarfile.open(tar_path, mode='r:gz') as data:
               data.extractall(where)

    extract_data('daily-stock','data') # this function call will extract json files
```

```
In [16]: import os, sys
         from glob import glob
         import pandas as pd
         import json

         here = os.getcwd() # get the current directory
         filenames = sorted(glob(os.path.join(here,'data', 'daily-stock', '*.json')))
```

```
In [17]: def read( fn ):
             with open(fn) as f:
                 return [json.loads(line) for line in f]

         def convert(data):
             df = pd.DataFrame(data)

             out_filename = fn[:-5] + '.h5'
             df.to_hdf(out_filename, os.path.join(here,'data'))
             return

         for fn in filenames:
             data = read( fn)
             convert(data)
```

```
/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/tables/path.py:155: NaturalNameWarning:
  check_attribute_name(name)
```

```
In [18]: %ls data/daily-stock/*.h5
```

```
data/daily-stock/aet.h5   data/daily-stock/bwa.h5  data/daily-stock/jpm.h5
data/daily-stock/afl.h5   data/daily-stock/ge.h5   data/daily-stock/luv.h5
data/daily-stock/aig.h5   data/daily-stock/hal.h5  data/daily-stock/met.h5
data/daily-stock/al.h5    data/daily-stock/hp.h5   data/daily-stock/pcg.h5
data/daily-stock/amgn.h5  data/daily-stock/hpq.h5  data/daily-stock/tgt.h5
data/daily-stock/avy.h5   data/daily-stock/ibm.h5  data/daily-stock/usb.h5
data/daily-stock/b.h5     data/daily-stock/jbl.h5  data/daily-stock/xom.h5
```

```
In [19]: @dask.delayed
         def read( fn ):
             " read json file "
             with open(fn) as f:
                 return [json.loads(line) for line in f]

         @dask.delayed
         def convert(data, fn):
             " convert json file to hdf5 file"
             df = pd.DataFrame(data)
             out_filename = fn[:-5] + '.h5'
             df.to_hdf(out_filename, '/data')
             return fn[:-5]
```

```
        results = []
        for filename in filenames:
            data = read(filename)
            results.append(convert(data, filename))
```

In [20]:
```
%time dask.compute(*results)
```

```
CPU times: user 8.96 s, sys: 1.21 s, total: 10.2 s
Wall time: 9.19 s
```

Out[20]:
```
('/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aet',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/afl',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aig',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/al',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/amgn',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/avy',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/b',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/bwa',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ge',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hal',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hp',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hpq',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ibm',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jbl',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jpm',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/luv',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/met',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/pcg',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/tgt',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/usb',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/xom')
```

Chapter 9

Dask bag

Dask proposes ”big data” collections with a small set of high-level primitives like `map`, `filter`, `groupby`, and `join`. With these common patterns we can often handle computations that are more complex than map, but are still structured.

- Dask-bag excels in processing data that can be represented as a sequence of arbitrary inputs (”messy” data)
- When you encounter a set of data with a format that does not enforce strict structure and datatypes.

Related Documentation

- Bag Documenation
- Bag API

```
In [1]: data = list(range(1,9))
        data
```

```
Out[1]: [1, 2, 3, 4, 5, 6, 7, 8]
```

```
In [2]: import dask.bag as db

        b = db.from_sequence(data)
```

```
In [3]: b.compute()  # Gather results back to local process
```

```
Out[3]: [1, 2, 3, 4, 5, 6, 7, 8]
```

```
In [4]: b.map(lambda x : x//2).compute() # compute length of each element and collect results
```

```
Out[4]: [0, 1, 1, 2, 2, 3, 3, 4]
```

```
In [5]: from time import sleep

        def slow_half( x):
            sleep(1)
            return x // 2

        res = b.map(slow_half)
        res
```

```
Out[5]: dask.bag<slow_half, npartitions=8>
```

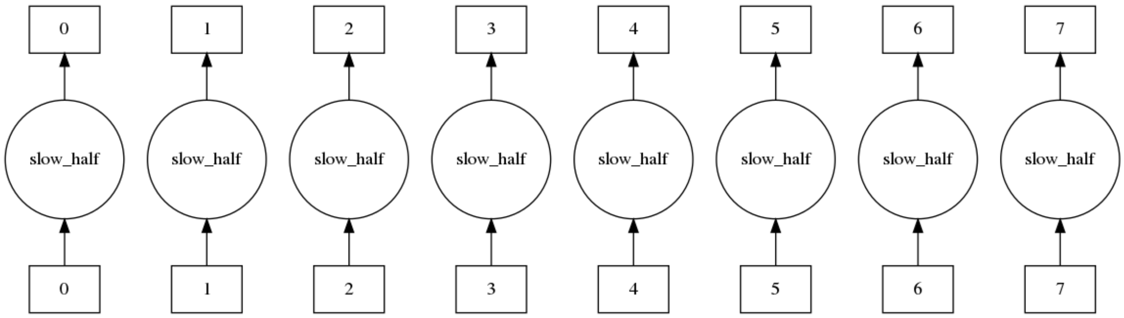
```
In [6]: %%time
        res.compute()
```

```
CPU times: user 15 ms, sys: 11.6 ms, total: 26.6 ms
Wall time: 4.39 s
```

```
Out[6]: [0, 1, 1, 2, 2, 3, 3, 4]
```

```
In [7]: res.visualize()
```

```
Out[7]:
```




```
In [8]: b.topk
```

```
Out[8]: <bound method Bag.topk of dask.bag<from_sequence, npartitions=8>>
```

```
In [9]: b.product(b).compute() # Cartesian product of each pair
        # of elements in two sequences (or the same sequence in this case)
```

```
Out[9]: [(1, 1),
         (1, 2),
         (1, 3),
         (1, 4),
         (1, 5),
         (1, 6),
         (1, 7),
         (1, 8),
         (2, 1),
         (2, 2),
         (2, 3),
         (2, 4),
         (2, 5),
         (2, 6),
         (2, 7),
         (2, 8),
         (3, 1),
         (3, 2),
         (3, 3),
         (3, 4),
         (3, 5),
         (3, 6),
         (3, 7),
         (3, 8),
         (4, 1),
```

```
 (4, 2),
 (4, 3),
 (4, 4),
 (4, 5),
 (4, 6),
 (4, 7),
 (4, 8),
 (5, 1),
 (5, 2),
 (5, 3),
 (5, 4),
 (5, 5),
 (5, 6),
 (5, 7),
 (5, 8),
 (6, 1),
 (6, 2),
 (6, 3),
 (6, 4),
 (6, 5),
 (6, 6),
 (6, 7),
 (6, 8),
 (7, 1),
 (7, 2),
 (7, 3),
 (7, 4),
 (7, 5),
 (7, 6),
 (7, 7),
 (7, 8),
 (8, 1),
 (8, 2),
 (8, 3),
 (8, 4),
 (8, 5),
 (8, 6),
 (8, 7),
 (8, 8)]
```

Chain operations to construct more complex computations

```
In [10]: (b.filter(lambda x: x % 2 > 0)
           .product(b)
           .filter( lambda v : v[0] % v[1] == 0 and v[0] != v[1])
           .compute())

Out[10]: [(3, 1), (5, 1), (7, 1)]

In [ ]:
```

9.1 Daily stock example

Let's use the bag interface to read the json files containing time series.

Each line is a JSON encoded dictionary with the following keys - timestamp: Day. - close: Stock value at the end of the day. - high: Highest value. - low: Lowest value. - open: Opening price.

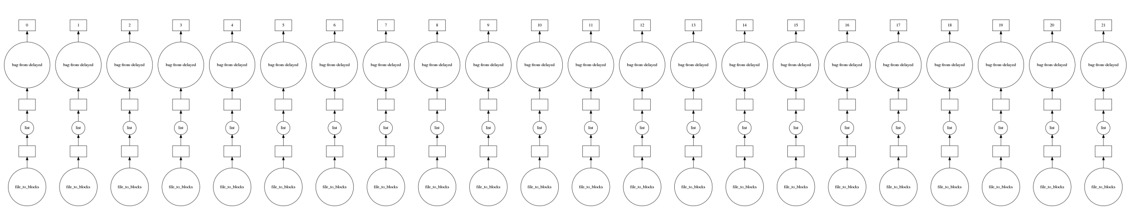
```
In [11]: # preparing data
         import os  # library to get directory and file paths
         import tarfile # this module makes possible to read and write tar archives

         def extract_data(name, where):
             datadir = os.path.join(where,name)
             if not os.path.exists(datadir):
                print("Extracting data...")
                tar_path = os.path.join(where, name+'.tgz')
                with tarfile.open(tar_path, mode='r:gz') as data:
                   data.extractall(where)

         extract_data('daily-stock','data') # this function call will extract json files
```

```
Extracting data…
```

```
In [12]: %ls data/daily-stock/*.json
```

```
data/daily-stock/aet.json   data/daily-stock/hpq.json
data/daily-stock/afl.json   data/daily-stock/ibm.json
data/daily-stock/aig.json   data/daily-stock/jbl.json
data/daily-stock/al.json    data/daily-stock/jpm.json
data/daily-stock/amgn.json  data/daily-stock/luv.json
data/daily-stock/avy.json   data/daily-stock/met.json
data/daily-stock/b.json     data/daily-stock/pcg.json
data/daily-stock/bwa.json   data/daily-stock/tgt.json
data/daily-stock/ge.json    data/daily-stock/usb.json
data/daily-stock/hal.json   data/daily-stock/xom.json
data/daily-stock/hp.json
```

```
In [13]: import dask.bag as db
         import json
         stocks = db.read_text('data/daily-stock/*.json')

In [14]: stocks.npartitions

Out[14]: 22

In [15]: stocks.visualize()

Out[15]:
```



```
In [16]: import json
         js = stocks.map(json.loads)
```

```
In [17]: import os, sys
         from glob import glob
         import pandas as pd
         import json

         here = os.getcwd() # get the current directory
         filenames = sorted(glob(os.path.join(here,'data', 'daily-stock', '*.json')))
         filenames
```

```
Out[17]: ['/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aet.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/afl.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aig.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/al.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/amgn.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/avy.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/b.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/bwa.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ge.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hal.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hp.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hpq.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ibm.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jbl.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jpm.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/luv.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/met.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/pcg.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/tgt.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/usb.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/xom.json']
```

```
In [18]: %rm data/daily-stock/*.h5
```

```
rm: cannot remove 'data/daily-stock/*.h5': No such file or directory
```

```
In [19]: from tqdm.notebook import tqdm
         for fn in tqdm(filenames):
             with open(fn) as f:
                 data = [json.loads(line) for line in f]

             df = pd.DataFrame(data)

             out_filename = fn[:-5] + '.h5'
             df.to_hdf(out_filename, '/data')
```

```
HBox(children=(FloatProgress(value=0.0, max=21.0), HTML(value='')))
```

```
In [20]: filenames = sorted(glob(os.path.join(here,'data', 'daily-stock', '*.h5')))
         filenames
```

```
Out[20]: ['/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aet.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/afl.h5',
```

```
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aig.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/al.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/amgn.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/avy.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/b.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/bwa.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ge.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hal.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hp.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hpq.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ibm.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jbl.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jpm.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/luv.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/met.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/pcg.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/tgt.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/usb.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/xom.h5']
```

9.1.1 Serial version

```
In [21]: %%time
         series = {}
         for fn in filenames:   # Simple map over filenames
             series[fn] = pd.read_hdf(fn)['close']

         results = {}

         for a in filenames:    # Doubly nested loop over the same collection
             for b in filenames:
                 if a != b:     # Filter out bad elements
                     results[a, b] = series[a].corr(series[b])  # Apply function

         ((a, b), corr) = max(results.items(), key=lambda kv: kv[1])  # Reduction
```

```
CPU times: user 1.01 s, sys: 47.9 ms, total: 1.06 s
Wall time: 1.06 s
```

```
In [22]: a, b, corr
```

```
Out[22]: ('/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aet.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/luv.h5',
          0.9413176064560879)
```

9.2 Dask.bag methods

We can construct most of the above computation with the following dask.bag methods:

- `collection.map(function)`: apply function to each element in collection
- `collection.product(collection)`: Create new collection with every pair of inputs
- `collection.filter(predicate)`: Keep only elements of colleciton that match the predicate function
- `collection.max()`: Compute maximum element

```
In [23]: %%time

         import dask.bag as db

         b = db.from_sequence(filenames)
         series = b.map(lambda fn: pd.read_hdf(fn)['close'])

         corr = (series.product(series)
                       .filter(lambda ab: not (ab[0] == ab[1]).all())
                       .map(lambda ab: ab[0].corr(ab[1])).max())
```

```
CPU times: user 6.14 ms, sys: 0 ns, total: 6.14 ms
Wall time: 5.94 ms
```

```
In [24]: %%time

         result = corr.compute()
```

```
CPU times: user 1.55 s, sys: 642 ms, total: 2.19 s
Wall time: 3.95 s
```

```
In [25]: result
```

```
Out[25]: 0.9413176064560879
```

9.2.1 Wordcount with Dask bag

```
In [26]: import lorem

         lorem.text()
```

```
Out[26]: 'Est voluptatem eius neque neque non voluptatem. Quiquia modi quaerat non sit non. Dolor adipi
```

```
In [27]: import lorem

         for i in range(20):
             with open(f"sample{i:02d}.txt","w") as f:
                 f.write(lorem.text())
```

```
In [28]: %ls *.txt
```

```
sample00.txt  sample04.txt  sample08.txt  sample12.txt  sample16.txt
sample01.txt  sample05.txt  sample09.txt  sample13.txt  sample17.txt
sample02.txt  sample06.txt  sample10.txt  sample14.txt  sample18.txt
sample03.txt  sample07.txt  sample11.txt  sample15.txt  sample19.txt
```

```
In [29]: import glob
         glob.glob('sample*.txt')
```

```
Out[29]: ['sample13.txt',
          'sample01.txt',
          'sample14.txt',
          'sample00.txt',
          'sample17.txt',
          'sample19.txt',
          'sample07.txt',
```

```
 'sample09.txt',
 'sample08.txt',
 'sample16.txt',
 'sample02.txt',
 'sample04.txt',
 'sample11.txt',
 'sample10.txt',
 'sample05.txt',
 'sample06.txt',
 'sample18.txt',
 'sample03.txt',
 'sample15.txt',
 'sample12.txt']
```

```
In [30]: import dask.bag as db
         import glob
         b = db.read_text(glob.glob('sample*.txt'))

         wordcount = (b.str.replace(".","")  # remove dots
                      .str.lower()           # lower text
                      .str.strip()           # remove \n and trailing spaces
                      .str.split()           # split into words
                      .flatten()             # chain all words lists
                      .frequencies()         # compute occurences
                      .topk(10, lambda x: x[1])) # sort and return top 10 words


         wordcount.compute() # Run all tasks and return result
```

```
Out[30]: [('quiquia', 165),
          ('dolore', 160),
          ('sit', 158),
          ('consectetur', 157),
          ('eius', 156),
          ('dolor', 154),
          ('ipsum', 152),
          ('est', 151),
          ('neque', 151),
          ('numquam', 149)]
```

9.3 Genome example

We will use a Dask bag to calculate the frequencies of sequences of five bases, and then sort the sequences into descending order ranked by their frequency.

- First we will define some functions to split the bases into sequences of a certain size

9.3.1 Exercise 9.1

- Implement a function `group_characters(line, n=5)` to group `n` characters together and return a iterator. `line` is a text line in genome.txt file.

```
>>> line = "abcdefghijklmno"
>>> for seq in group_character(line, 5):
        print(seq)
```

```
"abcde"
"efghi"
"klmno"
```

- Implement `group_and_split(line)`

```
>>> group_and_split('abcdefghijklmno')
['abcde', 'fghij', 'klmno']
```

- Use the dask bag to compute the frequencies of sequences of five bases.

```
In [31]: from string import ascii_lowercase as alphabet
         alphabet

Out[31]: 'abcdefghijklmnopqrstuvwxyz'

In [32]: def reverse(text):
             k = len(text)
             while k > 0:
                 k = k-1
                 yield text[k]

         reverse_alphabet = reverse(alphabet)
         print(*reverse_alphabet)

z y x w v u t s r q p o n m l k j i h g f e d c b a


In [33]: class Reverse:

             def __init__(self, data):
                 self.data = data
                 self.index = len(data)

             def __iter__(self):
                 return self

             def __next__(self):
                 self.index = self.index-1
                 if self.index < 0:
                     raise StopIteration
                 else:
                     return self.data[self.index]

In [34]: class Fibonacci:

             def __init__(self, n):
                 self.n = n
                 self.f0 = 0
                 self.f1 = 1

             def __iter__(self):
                 return self

             def __next__(self):
                 self.n = self.n - 1
                 if self.n < 0:
                     raise StopIteration
                 else:
```

```
                self.f0, self.f1 = self.f1, self.f0 + self.f1
            return self.f1

    print(*Fibonacci(7))
```

```
1 2 3 5 8 13 21
```

```
In [35]: for c in Reverse(alphabet):
             print(c, end="")
```

```
zyxwvutsrqponmlkjihgfedcba
```

```
In [36]:
         for c in reverse(alphabet):
             print(c, end="")
```

```
zyxwvutsrqponmlkjihgfedcba
```

```
In [37]: def group_character( line, n=5):
             bases = ''
             for i, b in enumerate(line):
                 bases += b
                 if (i+1) % n == 0:
                     yield bases
                     bases = ''
```

```
In [38]: line = "abcdefghijklmno"
         for seq in group_character(line, 5):
             print(seq)
```

```
abcde
fghij
klmno
```

```
In [39]: def group_and_split( line, n):
             return [seq for seq in group_character(line,n)]
```

```
In [40]: %ls data
```

```
daily-stock/     genome04.txt    monthly.land.90S.90N.df_1901-2000mean.dat.txt
daily-stock.tgz  genome05.txt    nucleotide-sample.txt
genome.txt       genome06.txt    nycflights.tar.gz
genome00.txt     genome07.txt    people.json
genome01.txt     irmar.csv       philadelphia-crime-data-2015-ytd.csv
genome02.txt     irmar.json
genome03.txt     latinbooks.tgz
```

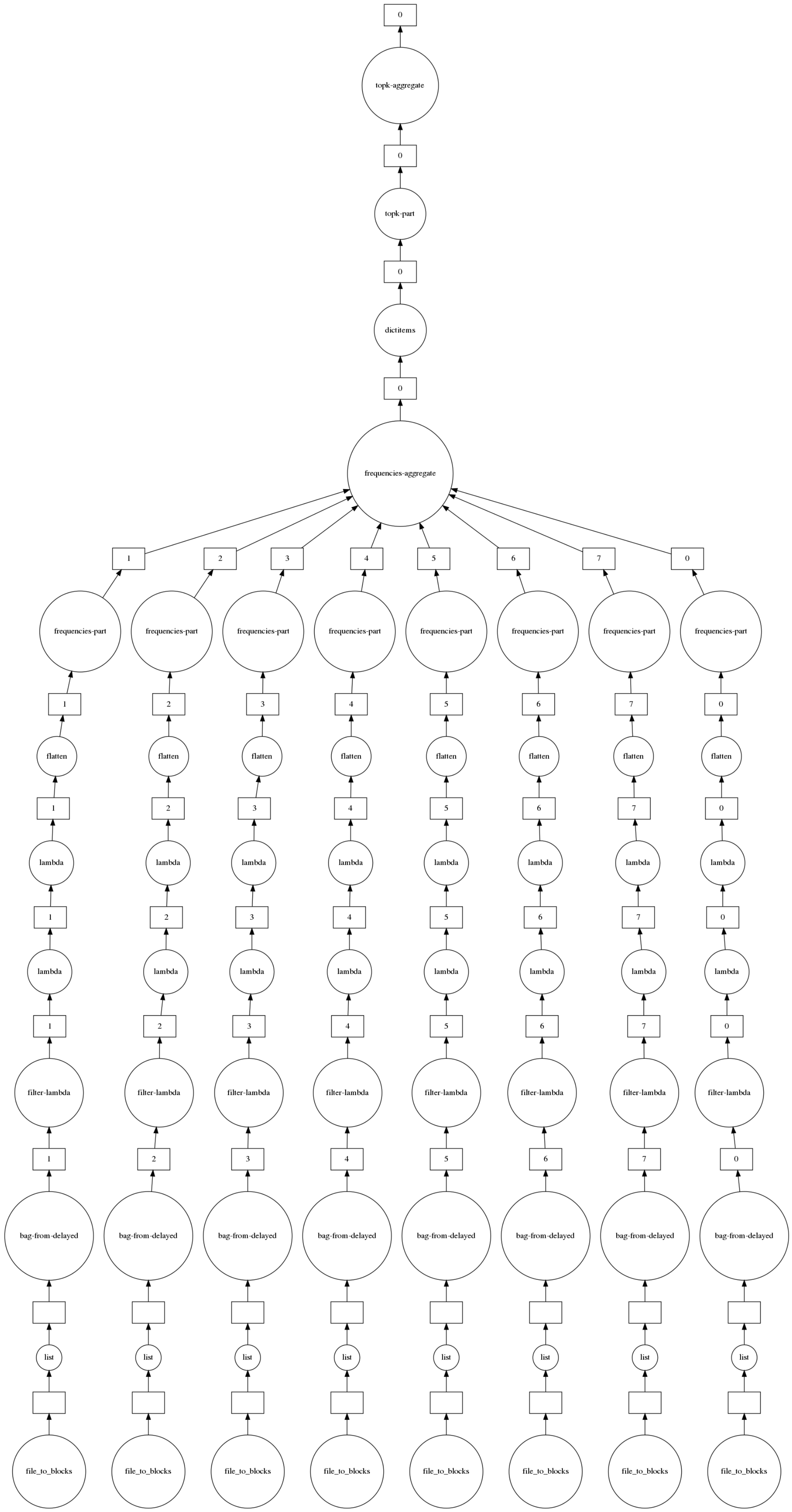
```
In [41]: import os
         from glob import glob

         data_path = os.path.join("data")
         with open(os.path.join(data_path,"genome.txt")) as g:
             data = g.read()
             for i in range(8):
                 file = os.path.join(data_path,f"genome{i:02d}.txt")
                 with open(file,"w") as f:
                     f.write(data)

         glob("data/genome0*.txt")
```

```
Out[41]: ['data/genome00.txt',
          'data/genome01.txt',
          'data/genome07.txt',
          'data/genome06.txt',
          'data/genome02.txt',
          'data/genome05.txt',
          'data/genome03.txt',
          'data/genome04.txt']
```

```
In [42]: from operator import itemgetter

         import dask.bag as db

         b = db.read_text("data/genome0*.txt")

         result = (b.filter(lambda line: not line.startswith(">"))
           .map(lambda line: line.strip())
           .map(lambda line : group_and_split(line,5))
           .flatten()
           .frequencies()
           .topk(10,lambda v : v[1]))
```

```
In [43]: result.visualize()
```

```
Out[43]:
```


0
topk-aggregate
0
topk-part
0
dictitems
0
frequencies-aggregate
1
2
3
4
5
6
7
0
frequencies-part
flatten
lambda
filter-lambda
bag-from-delayed
list
file_to_blocks

```
In [44]: result.compute()

Out[44]: [('CTGTG', 472),
          ('CCCAG', 440),
          ('CCTGG', 416),
          ('AAAAA', 392),
          ('TGCTG', 336),
          ('TGTGT', 328),
          ('CCACC', 312),
          ('GGCTG', 304),
          ('CACCA', 296),
          ('GGTGG', 296)]
```

9.3.2 Exercise 9.2

The FASTA file format is used to write several genome sequences.

- Create a function that can read a FASTA file and compute the frequencies for n = 5 of a given sequence.

9.3.3 Exercise 9.3

Write a program that uses the function implemented above to read several FASTA files stored in a Dask bag.

9.4 Some remarks about bag

- Higher level dask collections include functions for common patterns
- Move data to collection, construct lazy computation, trigger at the end
- Use Dask.bag (`product + map`) to handle nested for loop

Bags have the following known limitations

1. Bag operations tend to be slower than array/dataframe computations in the same way that Python tends to be slower than NumPy/Pandas
2. `Bag.groupby` is slow. You should try to use `Bag.foldby` if possible.
3. Check the API
4. `dask.dataframe` can be faster than `dask.bag`. But sometimes it is easier to load and clean messy data with a bag. We will see later how to transform a bag into a `dask.dataframe` with the to_dataframe method.

Chapter 10

Pandas Series

- Started by Wes MacKinney with a first release in 2011.
- Based on NumPy, it is the most used library for all things data.
- Motivated by the toolbox in R for manipulating data easily.
- A lot of names in Pandas come from R world.
- It is Open source (BSD)

https://pandas.pydata.org/

```
import pandas as pd
```

"*Pandas provides high-performance, easy-to-use data structures and data analysis tools in Python*"

- Self-describing data structures
- Data loaders to/from common file formats
- Plotting functions
- Basic statistical tools.

```
In [1]: %matplotlib inline
        %config InlineBackend.figure_format = 'retina'
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        sns.set()
        pd.set_option("display.max_rows", 8)
        plt.rcParams['figure.figsize'] = (9, 6)
```

10.1 Series

- A Series contains a one-dimensional array of data, *and* an associated sequence of labels called the *index*.
- The index can contain numeric, string, or date/time values.
- When the index is a time value, the series is a time series.
- The index must be the same length as the data.
- If no index is supplied it is automatically generated as `range(len(data))`.

```
In [2]: pd.Series([1,3,5,np.nan,6,8])

Out[2]: 0    1.0
        1    3.0
        2    5.0
        3    NaN
        4    6.0
        5    8.0
        dtype: float64

In [3]: pd.Series(index=pd.period_range('09/11/2017', '09/18/2017', freq="D"))

<ipython-input-3-579d6b723cc5>:1: DeprecationWarning: The default dtype for empty Series will be 'objec
  pd.Series(index=pd.period_range('09/11/2017', '09/18/2017', freq="D"))


Out[3]: 2017-09-11   NaN
        2017-09-12   NaN
        2017-09-13   NaN
        2017-09-14   NaN
        2017-09-15   NaN
        2017-09-16   NaN
        2017-09-17   NaN
        2017-09-18   NaN
        Freq: D, dtype: float64
```

10.1.1 Exercise

- Create a text with `lorem` and count word occurences with a `collection.Counter`. Put the result in a `dict`.

```
In [4]: from lorem import text
        from collections import Counter
        import operator

        c = Counter(filter(None,text().strip().replace('.','').replace('\n',' ').lower().split(' ')))
        result = dict(sorted(c.most_common(),key=operator.itemgetter(1),reverse=True))
        result

Out[4]: {'dolore': 14,
         'velit': 14,
         'quiquia': 12,
         'sed': 12,
         'adipisci': 11,
         'ut': 10,
         'etincidunt': 10,
         'modi': 10,
         'neque': 9,
         'non': 9,
```

```
 'eius': 9,
 'magnam': 8,
 'quisquam': 8,
 'aliquam': 8,
 'quaerat': 8,
 'est': 7,
 'dolorem': 7,
 'voluptatem': 7,
 'labore': 7,
 'dolor': 7,
 'tempora': 7,
 'amet': 6,
 'porro': 6,
 'sit': 5,
 'numquam': 5,
 'ipsum': 5,
 'consectetur': 4}
```

10.1.2 Exercise

- From the results create a Pandas series name latin_series with words in alphabetical order as index.

```
In [5]: df = pd.Series(result)
        df
```

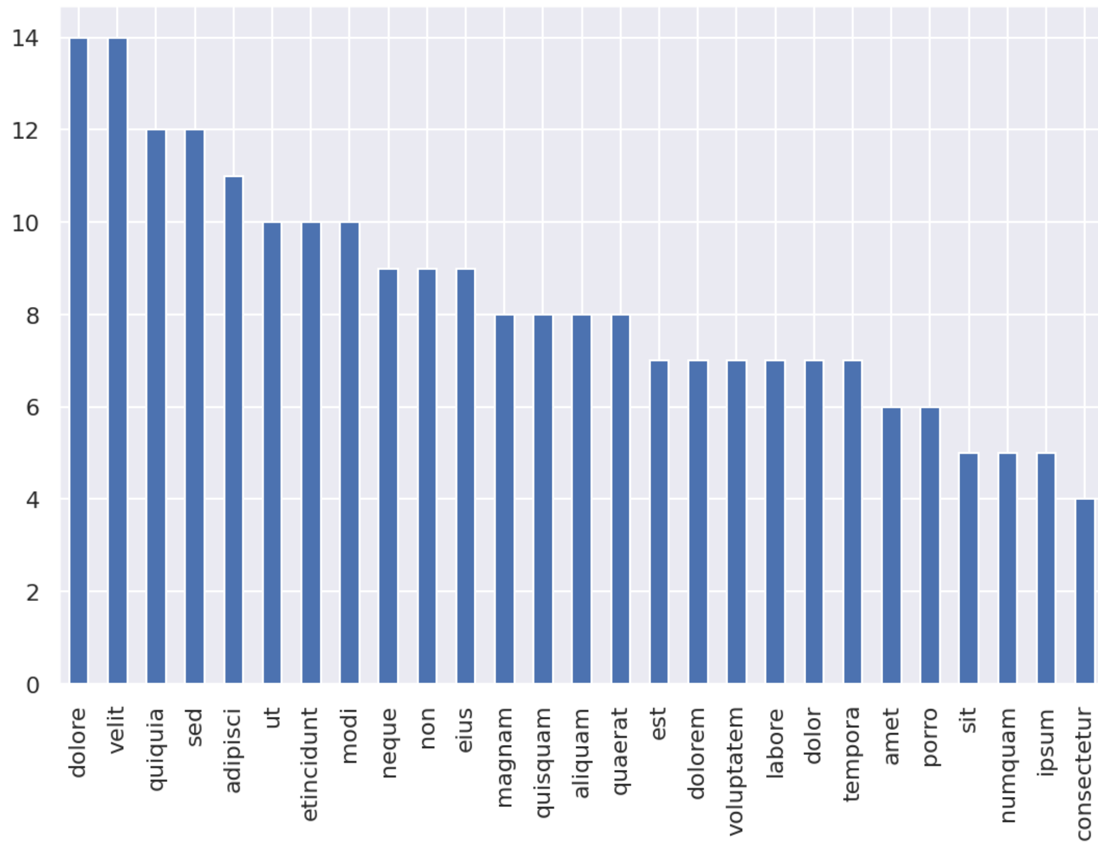
```
Out[5]: dolore         14
        velit          14
        quiquia        12
        sed            12
                       ..
        sit             5
        numquam         5
        ipsum           5
        consectetur     4
        Length: 27, dtype: int64
```

10.1.3 Exercise

- Plot the series using 'bar' kind.

```
In [6]: df.plot(kind='bar')
```

```
Out[6]: <AxesSubplot:>
```



10.1.4 Exercise

- Pandas provides explicit functions for indexing `loc` and `iloc`.
  - Use `loc` to display the number of occurrences of 'dolore'.
  - Use `iloc` to diplay the number of occurrences of the last word in index.

```
In [7]: df.loc['dolore']
```

```
Out[7]: 14
```

```
In [8]: df.iloc[-1]
```

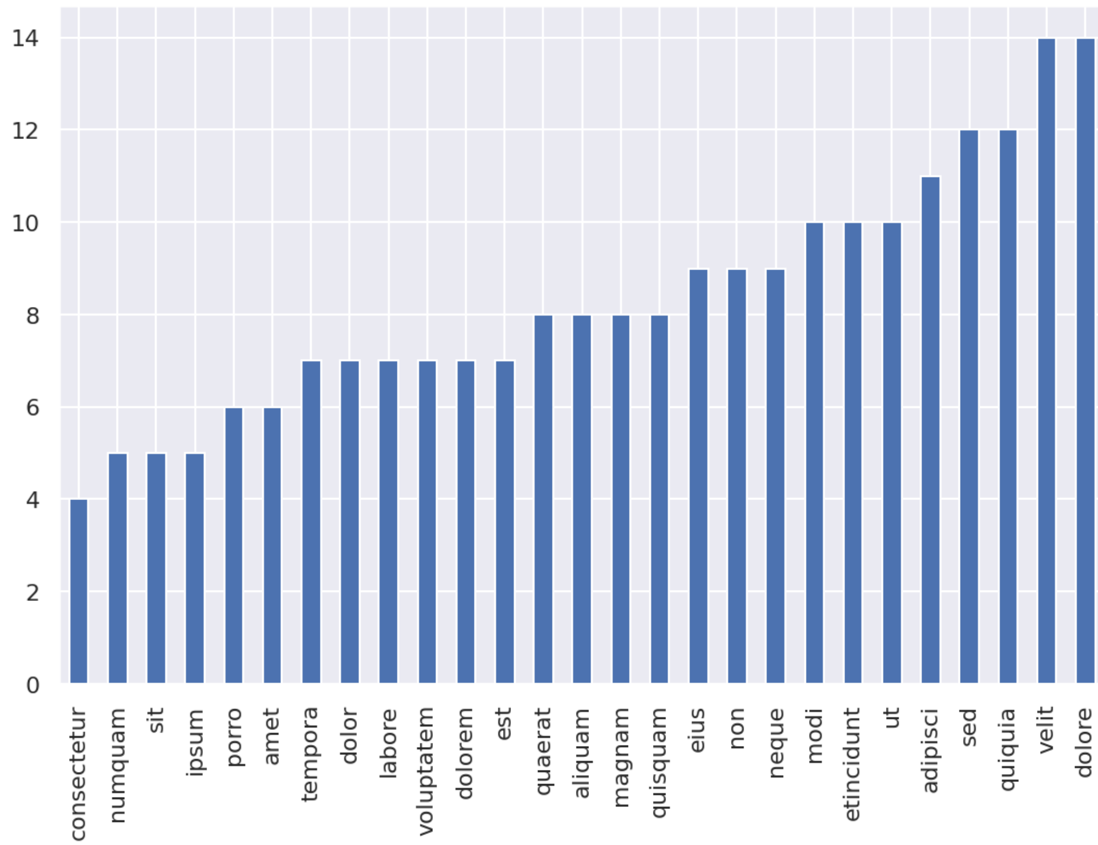
```
Out[8]: 4
```

10.1.5 Exercise

- Sort words by number of occurrences.
- Plot the Series.

```
In [9]: df = df.sort_values()
        df.plot(kind='bar')
```

```
Out[9]: <AxesSubplot:>
```



10.1.6 Full globe temperature between 1901 and 2000.

We read the text file and load the results in a pandas dataframe. In cells below you need to clean the data and convert the dataframe to a time series.

In [10]:
```
import os
here = os.getcwd()

filename = os.path.join(here,"data","monthly.land.90S.90N.df_1901-2000mean.dat.txt")

df = pd.read_table(filename, sep="\s+",
                   names=["year", "month", "mean temp"])
df
```

Out[10]:

	year	month	mean temp
0	1880	1	-0.0235
1	1880	2	-0.4936
2	1880	3	-0.6785
3	1880	4	-0.2829
…	…	…	…
1580	2011	9	-999.0000
1581	2011	10	-999.0000
1582	2011	11	-999.0000
1583	2011	12	-999.0000

```
[1584 rows x 3 columns]
```

10.1.7 Exercise

- Insert a third column with value one named "day" with `.insert`.
- convert df index to datetime with `pd.to_datetime` function.
- convert df to Series containing only "mean temp" column.

```
In [11]: df.insert(loc=2,column='day',value=np.ones(len(df)))
         df
```

```
Out[11]:       year  month  day  mean temp
         0     1880      1  1.0    -0.0235
         1     1880      2  1.0    -0.4936
         2     1880      3  1.0    -0.6785
         3     1880      4  1.0    -0.2829
         …     …     …  …          …
         1580  2011      9  1.0  -999.0000
         1581  2011     10  1.0  -999.0000
         1582  2011     11  1.0  -999.0000
         1583  2011     12  1.0  -999.0000

         [1584 rows x 4 columns]
```

```
In [12]: df.index = pd.to_datetime(df[['year','month','day']])
         df
```

```
Out[12]:             year  month  day  mean temp
         1880-01-01  1880      1  1.0    -0.0235
         1880-02-01  1880      2  1.0    -0.4936
         1880-03-01  1880      3  1.0    -0.6785
         1880-04-01  1880      4  1.0    -0.2829
         …           …     …  …          …
         2011-09-01  2011      9  1.0  -999.0000
         2011-10-01  2011     10  1.0  -999.0000
         2011-11-01  2011     11  1.0  -999.0000
         2011-12-01  2011     12  1.0  -999.0000

         [1584 rows x 4 columns]
```

```
In [13]: df = df['mean temp']
         df
```

```
Out[13]: 1880-01-01     -0.0235
         1880-02-01     -0.4936
         1880-03-01     -0.6785
         1880-04-01     -0.2829
                          …
         2011-09-01   -999.0000
         2011-10-01   -999.0000
         2011-11-01   -999.0000
         2011-12-01   -999.0000
         Name: mean temp, Length: 1584, dtype: float64
```

```
In [14]: type(df)
```

```
Out[14]: pandas.core.series.Series
```

10.1.8 Exercise

- Display the beginning of the file with `.head`.

```
In [15]: df.head()
```

```
Out[15]: 1880-01-01   -0.0235
         1880-02-01   -0.4936
         1880-03-01   -0.6785
         1880-04-01   -0.2829
         1880-05-01   -0.1261
         Name: mean temp, dtype: float64
```

10.1.9 Exercise

- Display the end of the file with `.tail`.

```
In [16]: df.tail()
```

```
Out[16]: 2011-08-01   -999.0
         2011-09-01   -999.0
         2011-10-01   -999.0
         2011-11-01   -999.0
         2011-12-01   -999.0
         Name: mean temp, dtype: float64
```

In the dataset, -999.00 was used to indicate that there was no value for that year.

10.1.10 Exercise

- Display values equal to -999 with `.values`.
- Replace the missing value (-999.000) by `np.nan`

```
In [17]: df[df.values == -999]
```

```
Out[17]: 2011-07-01   -999.0
         2011-08-01   -999.0
         2011-09-01   -999.0
         2011-10-01   -999.0
         2011-11-01   -999.0
         2011-12-01   -999.0
         Name: mean temp, dtype: float64
```

```
In [18]: df2 = df.copy()
         df2[df == -999.0] = np.nan  # For this indexing we need a copy
         df2.tail()
```

```
Out[18]: 2011-08-01   NaN
         2011-09-01   NaN
         2011-10-01   NaN
         2011-11-01   NaN
         2011-12-01   NaN
         Name: mean temp, dtype: float64
```

Once they have been converted to np.nan, missing values can be removed (dropped).

10.1.11 Exercise

- Remove missing values with `.dropna`.

```
In [19]: df = df2.dropna()
         df.tail()
```

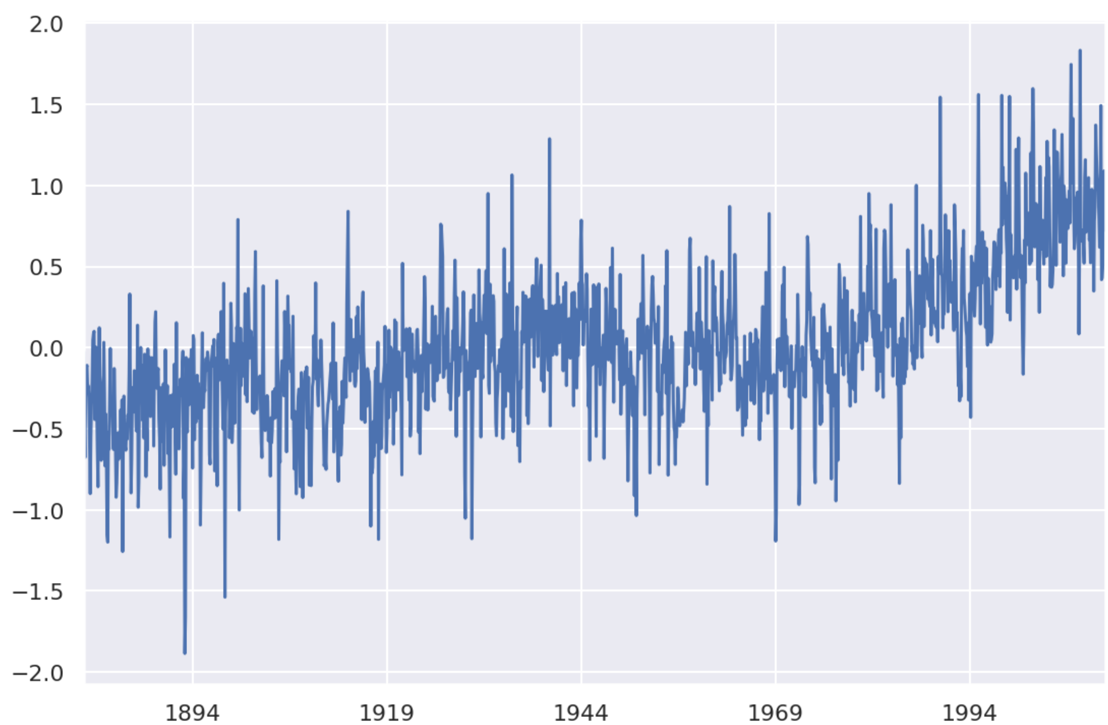
```
Out[19]: 2011-02-01    0.5113
         2011-03-01    0.8618
         2011-04-01    1.0897
         2011-05-01    0.7247
         2011-06-01    0.8550
         Name: mean temp, dtype: float64
```

10.1.12 Exercise

- Generate a basic visualization using `.plot`.

```
In [20]: df.plot()
```

```
Out[20]: <AxesSubplot:>
```



10.1.13 Exercise

Convert df index from timestamp to period is more meaningfull since it was measured and averaged over the month. Use `to_period` method.

```
In [21]: df = df.to_period('M')
         df
```

```
Out[21]: 1880-01   -0.0235
         1880-02   -0.4936
         1880-03   -0.6785
         1880-04   -0.2829
                     …
         2011-03    0.8618
         2011-04    1.0897
         2011-05    0.7247
         2011-06    0.8550
         Freq: M, Name: mean temp, Length: 1578, dtype: float64
```

10.2 Resampling

Series can be resample, downsample or upsample. - Frequencies can be specified as strings: ”us”, ”ms”, ”S”, ”T”, ”H”, ”D”, ”B”, ”W”, ”M”, ”A”, ”3min”, ”2h20”, ... - More aliases at http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases

10.2.1 Exercise

- With `resample` method, convert df Series to 10 year blocks:

```
In [22]: df.resample('10A').mean()
```

```
Out[22]: 1880   -0.386485
         1890   -0.316798
         1900   -0.256431
         1910   -0.247673
                   …
         1980    0.188519
         1990    0.463572
         2000    0.785452
         2010    0.884700
         Freq: 10A-DEC, Name: mean temp, Length: 14, dtype: float64
```

10.2.2 Saving Work

HDF5 is widely used and one of the most powerful file format to store binary data. It allows to store both Series and DataFrames.

```
In [23]: with pd.HDFStore("data/pandas_series.h5") as writer:
             df.to_hdf(writer, "/temperatures/full_globe")
```

10.2.3 Reloading data

```
In [24]: with pd.HDFStore("data/pandas_series.h5") as store:
             df = store["/temperatures/full_globe"]
```

Chapter 11

Pandas Dataframes

```
In [1]: %matplotlib inline
        %config InlineBackend.figure_format = 'retina'
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt

        pd.set_option("display.max_rows", 8)
        plt.rcParams['figure.figsize'] = (9, 6)
```

11.1 Create a DataFrame

```
In [2]: dates = pd.date_range('20130101', periods=6)
        pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))
```

Out[2]:

	A	B	C	D
2013-01-01	0.957754	0.123485	0.647874	-0.523251
2013-01-02	0.541358	-0.358241	1.126334	1.889239
2013-01-03	1.514614	-0.487712	-0.302023	-0.316535
2013-01-04	0.094265	1.210069	-2.605903	-2.194898
2013-01-05	1.539784	-1.027265	-0.924647	0.278974
2013-01-06	1.062518	-0.069908	-1.230252	0.271544

```
In [3]: pd.DataFrame({'A' : 1.,
                      'B' : pd.Timestamp('20130102'),
                      'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
                      'D' : np.arange(4,dtype='int32'),
                      'E' : pd.Categorical(["test","train","test","train"]),
                      'F' : 'foo' })
```

Out[3]:

	A	B	C	D	E	F
0	1.0	2013-01-02	1.0	0	test	foo
1	1.0	2013-01-02	1.0	1	train	foo
2	1.0	2013-01-02	1.0	2	test	foo
3	1.0	2013-01-02	1.0	3	train	foo

11.2 Load Data from CSV File

```
In [4]: url = "https://www.fun-mooc.fr/c4x/agrocampusouest/40001S03/asset/AnaDo_JeuDonnees_TemperatFrance.csv"
        french_cities = pd.read_csv(url, delimiter=";", encoding="latin1", index_col=0)
        french_cities
```

```
Out[4]:              Janv  Févr  Mars  Avri   Mai  Juin  juil  Août  Sept  Octo  Nove  \
        Bordeaux      5.6   6.6  10.3  12.8  15.8  19.3  20.9  21.0  18.6  13.8   9.1
        Brest         6.1   5.8   7.8   9.2  11.6  14.4  15.6  16.0  14.7  12.0   9.0
        Clermont      2.6   3.7   7.5  10.3  13.8  17.3  19.4  19.1  16.2  11.2   6.6
        Grenoble      1.5   3.2   7.7  10.6  14.5  17.8  20.1  19.5  16.7  11.4   6.5
        …           …   …   …   …   …   …   …   …   …   …   …
        Rennes        4.8   5.3   7.9  10.1  13.1  16.2  17.9  17.8  15.7  11.6   7.8
        Strasbourg    0.4   1.5   5.6   9.8  14.0  17.2  19.0  18.3  15.1   9.5   4.9
        Toulouse      4.7   5.6   9.2  11.6  14.9  18.7  20.9  20.9  18.3  13.3   8.6
        Vichy         2.4   3.4   7.1   9.9  13.6  17.1  19.3  18.8  16.0  11.0   6.6

                     Déce   Lati  Long   Moye  Ampl Région
        Bordeaux      6.2  44.50 -0.34  13.33  15.4     SO
        Brest         7.0  48.24 -4.29  10.77  10.2     NO
        Clermont      3.6  45.47  3.05  10.94  16.8     SE
        Grenoble      2.3  45.10  5.43  10.98  18.6     SE
        …           …    …    …    …    …     …
        Rennes        5.4  48.05 -1.41  11.13  13.1     NO
        Strasbourg    1.3  48.35  7.45   9.72  18.6     NE
        Toulouse      5.5  43.36  1.26  12.68  16.2     SO
        Vichy         3.4  46.08  3.26  10.72  16.9     SE

        [15 rows x 17 columns]
```

11.3 Viewing Data

```
In [5]: french_cities.head()
```

```
Out[5]:            Janv  Févr  Mars  Avri   Mai  Juin  juil  Août  Sept  Octo  Nove  \
        Bordeaux    5.6   6.6  10.3  12.8  15.8  19.3  20.9  21.0  18.6  13.8   9.1
        Brest       6.1   5.8   7.8   9.2  11.6  14.4  15.6  16.0  14.7  12.0   9.0
        Clermont    2.6   3.7   7.5  10.3  13.8  17.3  19.4  19.1  16.2  11.2   6.6
        Grenoble    1.5   3.2   7.7  10.6  14.5  17.8  20.1  19.5  16.7  11.4   6.5
        Lille       2.4   2.9   6.0   8.9  12.4  15.3  17.1  17.1  14.7  10.4   6.1

                   Déce   Lati  Long   Moye  Ampl Région
        Bordeaux    6.2  44.50 -0.34  13.33  15.4     SO
        Brest       7.0  48.24 -4.29  10.77  10.2     NO
        Clermont    3.6  45.47  3.05  10.94  16.8     SE
        Grenoble    2.3  45.10  5.43  10.98  18.6     SE
        Lille       3.5  50.38  3.04   9.73  14.7     NE
```

```
In [6]: french_cities.tail()
```

```
Out[6]:              Janv  Févr  Mars  Avri   Mai  Juin  juil  Août  Sept  Octo  Nove  \
        Paris         3.4   4.1   7.6  10.7  14.3  17.5  19.1  18.7  16.0  11.4   7.1
        Rennes        4.8   5.3   7.9  10.1  13.1  16.2  17.9  17.8  15.7  11.6   7.8
        Strasbourg    0.4   1.5   5.6   9.8  14.0  17.2  19.0  18.3  15.1   9.5   4.9
        Toulouse      4.7   5.6   9.2  11.6  14.9  18.7  20.9  20.9  18.3  13.3   8.6
        Vichy         2.4   3.4   7.1   9.9  13.6  17.1  19.3  18.8  16.0  11.0   6.6

                     Déce   Lati  Long   Moye  Ampl Région
        Paris         4.3  48.52  2.20  11.18  15.7     NE
        Rennes        5.4  48.05 -1.41  11.13  13.1     NO
```

```
Strasbourg   1.3  48.35  7.45   9.72  18.6     NE
Toulouse     5.5  43.36  1.26  12.68  16.2     SO
Vichy        3.4  46.08  3.26  10.72  16.9     SE
```

11.4 Index

```
In [7]: french_cities.index
```

```
Out[7]: Index(['Bordeaux', 'Brest', 'Clermont', 'Grenoble', 'Lille', 'Lyon',
               'Marseille', 'Montpellier', 'Nantes', 'Nice', 'Paris', 'Rennes',
               'Strasbourg', 'Toulouse', 'Vichy'],
              dtype='object')
```

We can rename an index by setting its name.

```
In [8]: french_cities.index.name = "City"
        french_cities.head()
```

```
Out[8]:           Janv  Févr  Mars  Avri   Mai  Juin  juil  Août  Sept  Octo  Nove  \
        City
        Bordeaux   5.6   6.6  10.3  12.8  15.8  19.3  20.9  21.0  18.6  13.8   9.1
        Brest      6.1   5.8   7.8   9.2  11.6  14.4  15.6  16.0  14.7  12.0   9.0
        Clermont   2.6   3.7   7.5  10.3  13.8  17.3  19.4  19.1  16.2  11.2   6.6
        Grenoble   1.5   3.2   7.7  10.6  14.5  17.8  20.1  19.5  16.7  11.4   6.5
        Lille      2.4   2.9   6.0   8.9  12.4  15.3  17.1  17.1  14.7  10.4   6.1

                  Déce   Lati  Long   Moye  Ampl Région
        City
        Bordeaux   6.2  44.50 -0.34  13.33  15.4     SO
        Brest      7.0  48.24 -4.29  10.77  10.2     NO
        Clermont   3.6  45.47  3.05  10.94  16.8     SE
        Grenoble   2.3  45.10  5.43  10.98  18.6     SE
        Lille      3.5  50.38  3.04   9.73  14.7     NE
```

11.4.1 Exercise: Rename DataFrame Months in English

```
In [9]: import locale
        import calendar

        locale.setlocale(locale.LC_ALL,'en_US')

        months = calendar.month_abbr
        print(*months)
```

```
        ---------------------------------------------------------------------------

        Error                                     Traceback (most recent call last)

        <ipython-input-9-e38d847f0b53> in <module>
          2 import calendar
          3
    ----> 4 locale.setlocale(locale.LC_ALL,'en_US')
          5
          6 months = calendar.month_abbr
```

```
/usr/share/miniconda3/envs/big-data/lib/python3.8/locale.py in setlocale(category, locale)
606         # convert to string
607         locale = normalize(_build_localename(locale))
--> 608     return _setlocale(category, locale)
609
610 def resetlocale(category=LC_ALL):


Error: unsupported locale setting
```

```
In [10]: french_cities.rename(
           columns={ old : new
                    for old, new in zip(french_cities.columns[:12], months[1:])
                   if old != new },
           inplace=True)
         french_cities.columns
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-10-39c4d54b672b> in <module>
  1 french_cities.rename(
  2   columns={ old : new
----> 3           for old, new in zip(french_cities.columns[:12], months[1:])
  4           if old != new },
  5   inplace=True)


NameError: name 'months' is not defined
```

```
In [11]: french_cities.rename(columns={'Moye':'Mean'}, inplace=True)
```

```
In [12]: french_cities
```

```
Out[12]:             Janv  Févr  Mars  Avri   Mai  Juin  juil  Août  Sept  Octo  Nove  \
         City
         Bordeaux     5.6   6.6  10.3  12.8  15.8  19.3  20.9  21.0  18.6  13.8   9.1
         Brest        6.1   5.8   7.8   9.2  11.6  14.4  15.6  16.0  14.7  12.0   9.0
         Clermont     2.6   3.7   7.5  10.3  13.8  17.3  19.4  19.1  16.2  11.2   6.6
         Grenoble     1.5   3.2   7.7  10.6  14.5  17.8  20.1  19.5  16.7  11.4   6.5
         …          …   …    …   …    …    …    …    …    …    …    …
         Rennes       4.8   5.3   7.9  10.1  13.1  16.2  17.9  17.8  15.7  11.6   7.8
         Strasbourg   0.4   1.5   5.6   9.8  14.0  17.2  19.0  18.3  15.1   9.5   4.9
         Toulouse     4.7   5.6   9.2  11.6  14.9  18.7  20.9  20.9  18.3  13.3   8.6
         Vichy        2.4   3.4   7.1   9.9  13.6  17.1  19.3  18.8  16.0  11.0   6.6

             Déce   Lati  Long   Mean  Ampl Région
         City
         Bordeaux     6.2  44.50 -0.34  13.33  15.4     SO
         Brest        7.0  48.24 -4.29  10.77  10.2     NO
         Clermont     3.6  45.47  3.05  10.94  16.8     SE
```

```
Grenoble     2.3  45.10  5.43  10.98  18.6     SE
…          …    …     …     …    …
Rennes       5.4  48.05 -1.41  11.13  13.1     NO
Strasbourg   1.3  48.35  7.45   9.72  18.6     NE
Toulouse     5.5  43.36  1.26  12.68  16.2     SO
Vichy        3.4  46.08  3.26  10.72  16.9     SE

[15 rows x 17 columns]
```

11.5 From a local or remote HTML file

We can download and extract data about mean sea level stations around the world from the PSMSL website.

```
In [13]: # Needs `lxml`, `beautifulSoup4` and `html5lib` python packages
         table_list = pd.read_html("http://www.psmsl.org/data/obtaining/")
```

```
In [14]: # there is 1 table on that page which contains metadata about the stations where
         # sea levels are recorded
         local_sea_level_stations = table_list[0]
         local_sea_level_stations
```

```
Out[14]:              Station Name    ID     Lat.      Lon.  GLOSS ID Country  \
         0                   BREST     1   48.383    -4.495     242.0     FRA
         1             SWINOUJSCIE     2   53.917    14.233       NaN     POL
         2               SHEERNESS     3   51.446     0.743       NaN     GBR
         3                HOLYHEAD     5   53.314    -4.620       NaN     GBR
         …                   …    …      …       …        …        …
         1544               SUVA-B  2356 -18.133   178.428       NaN     FJI
         1545   SYDNEY PORT JACKSON  2358 -33.826   151.259       NaN     AUS
         1546                 ARKO  2359  58.484    16.961       NaN     SWE
         1547            UDDEVALLA  2360  58.348    11.895       NaN     SWE

                    Date  Coastline  Station
         0     07/08/2019        190       91
         1     19/10/2001        110       92
         2     06/06/2019        170      101
         3     06/06/2019        170      191
         …          …          …       …
         1544  28/01/2020        742       14
         1545  13/06/2019        680      138
         1546  12/09/2019         50      112
         1547  12/09/2019         50       22

         [1548 rows x 9 columns]
```

11.6 Indexing on DataFrames

```
In [15]: french_cities['Lati']  # DF [] accesses columns (Series)
```

```
Out[15]: City
         Bordeaux      44.50
         Brest         48.24
         Clermont      45.47
         Grenoble      45.10
```

```
              …
Rennes        48.05
Strasbourg    48.35
Toulouse      43.36
Vichy         46.08
Name: Lati, Length: 15, dtype: float64
```

.loc and .iloc allow to access individual values, slices or masked selections:

```
In [16]: french_cities.loc['Rennes', "Sep"]
```

```
    ---------------------------------------------------------------------------

    KeyError                                  Traceback (most recent call last)

    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexes/base.py in g
   2888             try:
-> 2889                 return self._engine.get_loc(casted_key)
   2890             except KeyError as err:


    pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()


    pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()


    pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()


    pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()


    KeyError: 'Sep'


The above exception was the direct cause of the following exception:


    KeyError                                  Traceback (most recent call last)

    <ipython-input-16-766b5d3b5de4> in <module>
----> 1 french_cities.loc['Rennes', "Sep"]


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in __get
    871                     # AttributeError for IntervalTree get_value
    872                     pass
--> 873             return self._getitem_tuple(key)
    874         else:
    875             # we by definition only have the 0th axis


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in _geti
```

```
   1042     def _getitem_tuple(self, tup: Tuple):
   1043         try:
-> 1044             return self._getitem_lowerdim(tup)
   1045         except IndexingError:
   1046             pass


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in _geti
    808                     return section
    809                 # This is an elided recursive call to iloc/loc
--> 810                 return getattr(section, self.name)[new_key]
    811
    812         raise IndexingError("not applicable")


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in __get
    877
    878             maybe_callable = com.apply_if_callable(key, self.obj)
--> 879             return self._getitem_axis(maybe_callable, axis=axis)
    880
    881     def _is_scalar_access(self, key: Tuple):


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in _geti
   1108         # fall thru to straight lookup
   1109         self._validate_key(key, axis)
-> 1110         return self._get_label(key, axis=axis)
   1111
   1112     def _get_slice_axis(self, slice_obj: slice, axis: int):


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in _get_
   1057     def _get_label(self, label, axis: int):
   1058         # GH#5667 this will fail if the label is not present in the axis.
-> 1059         return self.obj.xs(label, axis=axis)
   1060
   1061     def _handle_lowerdim_multi_index_axis0(self, tup: Tuple):


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/generic.py in xs(sel
   3480             loc, new_index = self.index.get_loc_level(key, drop_level=drop_level)
   3481         else:
-> 3482             loc = self.index.get_loc(key)
   3483
   3484             if isinstance(loc, np.ndarray):


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexes/base.py in g
   2889                 return self._engine.get_loc(casted_key)
   2890             except KeyError as err:
-> 2891                 raise KeyError(key) from err
   2892
   2893         if tolerance is not None:
```

```
KeyError: 'Sep'
```

```
In [17]: french_cities.loc['Rennes', ["Sep", "Dec"]]
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<ipython-input-17-988685453654> in <module>
----> 1 french_cities.loc['Rennes', ["Sep", "Dec"]]


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in __get
    871                     # AttributeError for IntervalTree get_value
    872                     pass
--> 873             return self._getitem_tuple(key)
    874         else:
    875             # we by definition only have the 0th axis


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in _geti
   1042     def _getitem_tuple(self, tup: Tuple):
   1043         try:
-> 1044             return self._getitem_lowerdim(tup)
   1045         except IndexingError:
   1046             pass


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in _geti
    808                     return section
    809                 # This is an elided recursive call to iloc/loc
--> 810                 return getattr(section, self.name)[new_key]
    811
    812         raise IndexingError("not applicable")


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in __get
    877
    878             maybe_callable = com.apply_if_callable(key, self.obj)
--> 879             return self._getitem_axis(maybe_callable, axis=axis)
    880
    881     def _is_scalar_access(self, key: Tuple):


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in _geti
   1097                     raise ValueError("Cannot index with multidimensional key")
   1098
-> 1099                 return self._getitem_iterable(key, axis=axis)
   1100
   1101             # nested tuple slicing
```

```
/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in _geti
   1035
   1036         # A collection of keys
-> 1037         keyarr, indexer = self._get_listlike_indexer(key, axis, raise_missing=False)
   1038         return self.obj._reindex_with_indexers(
   1039             {axis: [keyarr, indexer]}, copy=True, allow_dups=True


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in _get_
   1252             keyarr, indexer, new_indexer = ax._reindex_non_unique(keyarr)
   1253
-> 1254         self._validate_read_indexer(keyarr, indexer, axis, raise_missing=raise_missing)
   1255         return keyarr, indexer
   1256


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in _vali
   1296             if missing == len(indexer):
   1297                 axis_name = self.obj._get_axis_name(axis)
-> 1298                 raise KeyError(f"None of [{key}] are in the [{axis_name}]")
   1299
   1300             # We (temporarily) allow for some missing keys with .loc, except in


KeyError: "None of [Index(['Sep', 'Dec'], dtype='object')] are in the [index]"
```

```
In [18]: french_cities.loc['Rennes', "Sep":"Dec"]
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexes/base.py in g
   2888             try:
-> 2889                 return self._engine.get_loc(casted_key)
   2890             except KeyError as err:


pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()


pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()


pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()


pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()


KeyError: 'Sep'
```

```
The above exception was the direct cause of the following exception:


    KeyError                                  Traceback (most recent call last)

    <ipython-input-18-9347c35b6c44> in <module>
----> 1 french_cities.loc['Rennes', "Sep":"Dec"]


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in __get
    871                     # AttributeError for IntervalTree get_value
    872                     pass
--> 873             return self._getitem_tuple(key)
    874         else:
    875             # we by definition only have the 0th axis


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in _geti
   1042     def _getitem_tuple(self, tup: Tuple):
   1043         try:
-> 1044             return self._getitem_lowerdim(tup)
   1045         except IndexingError:
   1046             pass


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in _geti
    808                     return section
    809                 # This is an elided recursive call to iloc/loc
--> 810                 return getattr(section, self.name)[new_key]
    811 
    812         raise IndexingError("not applicable")


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in __get
    877 
    878             maybe_callable = com.apply_if_callable(key, self.obj)
--> 879             return self._getitem_axis(maybe_callable, axis=axis)
    880 
    881     def _is_scalar_access(self, key: Tuple):


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in _geti
   1086         if isinstance(key, slice):
   1087             self._validate_key(key, axis)
-> 1088             return self._get_slice_axis(key, axis=axis)
   1089         elif com.is_bool_indexer(key):
   1090             return self._getbool_axis(key, axis=axis)


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexing.py in _get_
   1120 
   1121         labels = obj._get_axis(axis)
-> 1122         indexer = labels.slice_indexer(
   1123             slice_obj.start, slice_obj.stop, slice_obj.step, kind="loc"
```

```
   1124         )


 /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexes/base.py in s
   4958         slice(1, 3, None)
   4959         """
-> 4960         start_slice, end_slice = self.slice_locs(start, end, step=step, kind=kind)
   4961
   4962         # return a slice


 /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexes/base.py in s
   5159         start_slice = None
   5160         if start is not None:
-> 5161             start_slice = self.get_slice_bound(start, "left", kind)
   5162         if start_slice is None:
   5163             start_slice = 0


 /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexes/base.py in g
   5081             except ValueError:
   5082                 # raise the original KeyError
-> 5083                 raise err
   5084
   5085         if isinstance(slc, np.ndarray):


 /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexes/base.py in g
   5075         # we need to look up the label
   5076         try:
-> 5077             slc = self.get_loc(label)
   5078         except KeyError as err:
   5079             try:


 /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexes/base.py in g
   2889                 return self._engine.get_loc(casted_key)
   2890             except KeyError as err:
-> 2891                 raise KeyError(key) from err
   2892
   2893         if tolerance is not None:


 KeyError: 'Sep'
```

11.7 Masking

```
In [19]: mask = [True, False] * 6 + 5 * [False]
         print(french_cities.iloc[:, mask])

            Janv  Mars   Mai  juil  Sept  Nove
City
Bordeaux     5.6  10.3  15.8  20.9  18.6   9.1
```

```
Brest        6.1   7.8  11.6  15.6  14.7   9.0
Clermont     2.6   7.5  13.8  19.4  16.2   6.6
Grenoble     1.5   7.7  14.5  20.1  16.7   6.5
…          …   …    …   …    …    …
Rennes       4.8   7.9  13.1  17.9  15.7   7.8
Strasbourg   0.4   5.6  14.0  19.0  15.1   4.9
Toulouse     4.7   9.2  14.9  20.9  18.3   8.6
Vichy        2.4   7.1  13.6  19.3  16.0   6.6

[15 rows x 6 columns]
```

```
In [20]: print(french_cities.loc["Rennes", mask])
```

```
Janv     4.8
Mars     7.9
Mai     13.1
juil    17.9
Sept    15.7
Nove     7.8
Name: Rennes, dtype: object
```

11.8 New column

```
In [21]: french_cities["std"] = french_cities.iloc[:,:12].std(axis=1)
         french_cities
```

```
Out[21]:             Janv  Févr  Mars  Avri   Mai  Juin  juil  Août  Sept  Octo  Nove  \
         City
         Bordeaux     5.6   6.6  10.3  12.8  15.8  19.3  20.9  21.0  18.6  13.8   9.1
         Brest        6.1   5.8   7.8   9.2  11.6  14.4  15.6  16.0  14.7  12.0   9.0
         Clermont     2.6   3.7   7.5  10.3  13.8  17.3  19.4  19.1  16.2  11.2   6.6
         Grenoble     1.5   3.2   7.7  10.6  14.5  17.8  20.1  19.5  16.7  11.4   6.5
         …          …   …    …   …    …    …    …    …    …    …    …    …
         Rennes       4.8   5.3   7.9  10.1  13.1  16.2  17.9  17.8  15.7  11.6   7.8
         Strasbourg   0.4   1.5   5.6   9.8  14.0  17.2  19.0  18.3  15.1   9.5   4.9
         Toulouse     4.7   5.6   9.2  11.6  14.9  18.7  20.9  20.9  18.3  13.3   8.6
         Vichy        2.4   3.4   7.1   9.9  13.6  17.1  19.3  18.8  16.0  11.0   6.6

             Déce   Lati  Long   Mean  Ampl Région       std
         City
         Bordeaux     6.2  44.50 -0.34  13.33  15.4     SO  5.792681
         Brest        7.0  48.24 -4.29  10.77  10.2     NO  3.773673
         Clermont     3.6  45.47  3.05  10.94  16.8     SE  6.189795
         Grenoble     2.3  45.10  5.43  10.98  18.6     SE  6.770771
         …          …    …    …      …    …      …       …
         Rennes       5.4  48.05 -1.41  11.13  13.1     NO  4.958800
         Strasbourg   1.3  48.35  7.45   9.72  18.6     NE  6.931723
         Toulouse     5.5  43.36  1.26  12.68  16.2     SO  6.056977
         Vichy        3.4  46.08  3.26  10.72  16.9     SE  6.201148

         [15 rows x 18 columns]
```

```
In [22]: french_cities = french_cities.drop("std", axis=1) # remove this new column
```

In [23]:
```
french_cities
```

Out[23]:
```
            Janv  Févr  Mars  Avri   Mai  Juin  juil  Août  Sept  Octo  Nove  \
City
Bordeaux     5.6   6.6  10.3  12.8  15.8  19.3  20.9  21.0  18.6  13.8   9.1
Brest        6.1   5.8   7.8   9.2  11.6  14.4  15.6  16.0  14.7  12.0   9.0
Clermont     2.6   3.7   7.5  10.3  13.8  17.3  19.4  19.1  16.2  11.2   6.6
Grenoble     1.5   3.2   7.7  10.6  14.5  17.8  20.1  19.5  16.7  11.4   6.5
…          …   …   …   …   …   …   …   …   …   …   …
Rennes       4.8   5.3   7.9  10.1  13.1  16.2  17.9  17.8  15.7  11.6   7.8
Strasbourg   0.4   1.5   5.6   9.8  14.0  17.2  19.0  18.3  15.1   9.5   4.9
Toulouse     4.7   5.6   9.2  11.6  14.9  18.7  20.9  20.9  18.3  13.3   8.6
Vichy        2.4   3.4   7.1   9.9  13.6  17.1  19.3  18.8  16.0  11.0   6.6

            Déce   Lati  Long   Mean  Ampl Région
City
Bordeaux     6.2  44.50 -0.34  13.33  15.4     SO
Brest        7.0  48.24 -4.29  10.77  10.2     NO
Clermont     3.6  45.47  3.05  10.94  16.8     SE
Grenoble     2.3  45.10  5.43  10.98  18.6     SE
…          …    …    …    …    …    …
Rennes       5.4  48.05 -1.41  11.13  13.1     NO
Strasbourg   1.3  48.35  7.45   9.72  18.6     NE
Toulouse     5.5  43.36  1.26  12.68  16.2     SO
Vichy        3.4  46.08  3.26  10.72  16.9     SE

[15 rows x 17 columns]
```

11.9 Modifying a dataframe with multiple indexing

In [24]:
```
# french_cities['Rennes']['Sep'] = 25 # It does not works and breaks the DataFrame
french_cities.loc['Rennes']['Sep'] # = 25 is the right way to do it
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexes/base.py in g
   2888             try:
-> 2889                 return self._engine.get_loc(casted_key)
   2890             except KeyError as err:


pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()


pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()


pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()


pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()
```

```
    KeyError: 'Sep'


The above exception was the direct cause of the following exception:


    KeyError                                  Traceback (most recent call last)

    <ipython-input-24-c68575f89e60> in <module>
      1 # french_cities['Rennes']['Sep'] = 25 # It does not works and breaks the DataFrame
----> 2 french_cities.loc['Rennes']['Sep'] # = 25 is the right way to do it


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/series.py in __geti
    880
    881         elif key_is_scalar:
--> 882             return self._get_value(key)
    883
    884         if (


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/series.py in _get_va
    989
    990         # Similar to Index.get_value, but we do not fall back to positional
--> 991         loc = self.index.get_loc(label)
    992         return self.index._get_values_for_loc(self, loc, label)
    993


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/core/indexes/base.py in g
   2889                 return self._engine.get_loc(casted_key)
   2890             except KeyError as err:
-> 2891                 raise KeyError(key) from err
   2892
   2893         if tolerance is not None:


    KeyError: 'Sep'
```

```
In [25]: french_cities
```

```
Out[25]:             Janv  Févr  Mars  Avri   Mai  Juin  juil  Août  Sept  Octo  Nove  \
          City
          Bordeaux     5.6   6.6  10.3  12.8  15.8  19.3  20.9  21.0  18.6  13.8   9.1
          Brest        6.1   5.8   7.8   9.2  11.6  14.4  15.6  16.0  14.7  12.0   9.0
          Clermont     2.6   3.7   7.5  10.3  13.8  17.3  19.4  19.1  16.2  11.2   6.6
          Grenoble     1.5   3.2   7.7  10.6  14.5  17.8  20.1  19.5  16.7  11.4   6.5
          …          …   …   …   …   …   …   …   …   …   …   …
          Rennes       4.8   5.3   7.9  10.1  13.1  16.2  17.9  17.8  15.7  11.6   7.8
          Strasbourg   0.4   1.5   5.6   9.8  14.0  17.2  19.0  18.3  15.1   9.5   4.9
          Toulouse     4.7   5.6   9.2  11.6  14.9  18.7  20.9  20.9  18.3  13.3   8.6
          Vichy        2.4   3.4   7.1   9.9  13.6  17.1  19.3  18.8  16.0  11.0   6.6
```

```
              Déce   Lati  Long   Mean  Ampl Région
City
Bordeaux       6.2  44.50 -0.34  13.33  15.4      SO
Brest          7.0  48.24 -4.29  10.77  10.2      NO
Clermont       3.6  45.47  3.05  10.94  16.8      SE
Grenoble       2.3  45.10  5.43  10.98  18.6      SE
…            …    …     …     …    …     …
Rennes         5.4  48.05 -1.41  11.13  13.1      NO
Strasbourg     1.3  48.35  7.45   9.72  18.6      NE
Toulouse       5.5  43.36  1.26  12.68  16.2      SO
Vichy          3.4  46.08  3.26  10.72  16.9      SE

[15 rows x 17 columns]
```

11.10 Transforming datasets

```
In [26]: french_cities['Mean'].min(), french_cities['Ampl'].max()
```

```
Out[26]: (9.72, 18.6)
```

11.11 Apply

Let's convert the temperature mean from Celsius to Fahrenheit degree.

```
In [27]: fahrenheit = lambda T: T*9/5+32
         french_cities['Mean'].apply(fahrenheit)
```

```
Out[27]: City
         Bordeaux      55.994
         Brest         51.386
         Clermont      51.692
         Grenoble      51.764
                         …
         Rennes        52.034
         Strasbourg    49.496
         Toulouse      54.824
         Vichy         51.296
         Name: Mean, Length: 15, dtype: float64
```

11.12 Sort

```
In [28]: french_cities.sort_values(by='Lati')
```

```
Out[28]:              Janv  Févr  Mars  Avri   Mai  Juin  juil  Août  Sept  Octo  Nove  \
         City
         Marseille     5.5   6.6  10.0  13.0  16.8  20.8  23.3  22.8  19.9  15.0  10.2
         Montpellier   5.6   6.7   9.9  12.8  16.2  20.1  22.7  22.3  19.3  14.6  10.0
         Toulouse      4.7   5.6   9.2  11.6  14.9  18.7  20.9  20.9  18.3  13.3   8.6
         Nice          7.5   8.5  10.8  13.3  16.7  20.1  22.7  22.5  20.3  16.0  11.5
         …           …   …    …   …    …    …   …    …    …    …    …
         Brest         6.1   5.8   7.8   9.2  11.6  14.4  15.6  16.0  14.7  12.0   9.0
         Strasbourg    0.4   1.5   5.6   9.8  14.0  17.2  19.0  18.3  15.1   9.5   4.9
         Paris         3.4   4.1   7.6  10.7  14.3  17.5  19.1  18.7  16.0  11.4   7.1
```

```
Lille          2.4   2.9   6.0   8.9  12.4  15.3  17.1  17.1  14.7  10.4   6.1

             Déce   Lati  Long   Mean  Ampl Région
City
Marseille      6.9  43.18  5.24  14.23  17.8     SE
Montpellier    6.5  43.36  3.53  13.89  17.1     SE
Toulouse       5.5  43.36  1.26  12.68  16.2     SO
Nice           8.2  43.42  7.15  14.84  15.2     SE
…            …    …     …     …    …     …
Brest          7.0  48.24 -4.29  10.77  10.2     NO
Strasbourg     1.3  48.35  7.45   9.72  18.6     NE
Paris          4.3  48.52  2.20  11.18  15.7     NE
Lille          3.5  50.38  3.04   9.73  14.7     NE

[15 rows x 17 columns]
```

```
In [29]: french_cities = french_cities.sort_values(by='Lati',ascending=False)
         french_cities
```

```
Out[29]:             Janv  Févr  Mars  Avri   Mai  Juin  juil  Août  Sept  Octo  Nove  \
City
Lille          2.4   2.9   6.0   8.9  12.4  15.3  17.1  17.1  14.7  10.4   6.1
Paris          3.4   4.1   7.6  10.7  14.3  17.5  19.1  18.7  16.0  11.4   7.1
Strasbourg     0.4   1.5   5.6   9.8  14.0  17.2  19.0  18.3  15.1   9.5   4.9
Brest          6.1   5.8   7.8   9.2  11.6  14.4  15.6  16.0  14.7  12.0   9.0
…            …    …    …    …    …    …    …    …    …    …    …
Nice           7.5   8.5  10.8  13.3  16.7  20.1  22.7  22.5  20.3  16.0  11.5
Montpellier    5.6   6.7   9.9  12.8  16.2  20.1  22.7  22.3  19.3  14.6  10.0
Toulouse       4.7   5.6   9.2  11.6  14.9  18.7  20.9  20.9  18.3  13.3   8.6
Marseille      5.5   6.6  10.0  13.0  16.8  20.8  23.3  22.8  19.9  15.0  10.2

             Déce   Lati  Long   Mean  Ampl Région
City
Lille          3.5  50.38  3.04   9.73  14.7     NE
Paris          4.3  48.52  2.20  11.18  15.7     NE
Strasbourg     1.3  48.35  7.45   9.72  18.6     NE
Brest          7.0  48.24 -4.29  10.77  10.2     NO
…            …    …     …     …    …     …
Nice           8.2  43.42  7.15  14.84  15.2     SE
Montpellier    6.5  43.36  3.53  13.89  17.1     SE
Toulouse       5.5  43.36  1.26  12.68  16.2     SO
Marseille      6.9  43.18  5.24  14.23  17.8     SE

[15 rows x 17 columns]
```

11.13 Stack and unstack

Instead of seeing the months along the axis 1, and the cities along the axis 0, let's try to convert these into an outer and an inner axis along only 1 time dimension.

```
In [30]: pd.set_option("display.max_rows", 20)
         unstacked = french_cities.iloc[:,:12].unstack()
         unstacked
```

```
Out[30]:       City
         Janv  Lille          2.4
```

```
       Paris          3.4
       Strasbourg     0.4
       Brest          6.1
       Rennes         4.8
                     …
 Déce  Bordeaux       6.2
       Nice           8.2
       Montpellier    6.5
       Toulouse       5.5
       Marseille      6.9
 Length: 180, dtype: float64
```

```
In [31]: type(unstacked)
```

```
Out[31]: pandas.core.series.Series
```

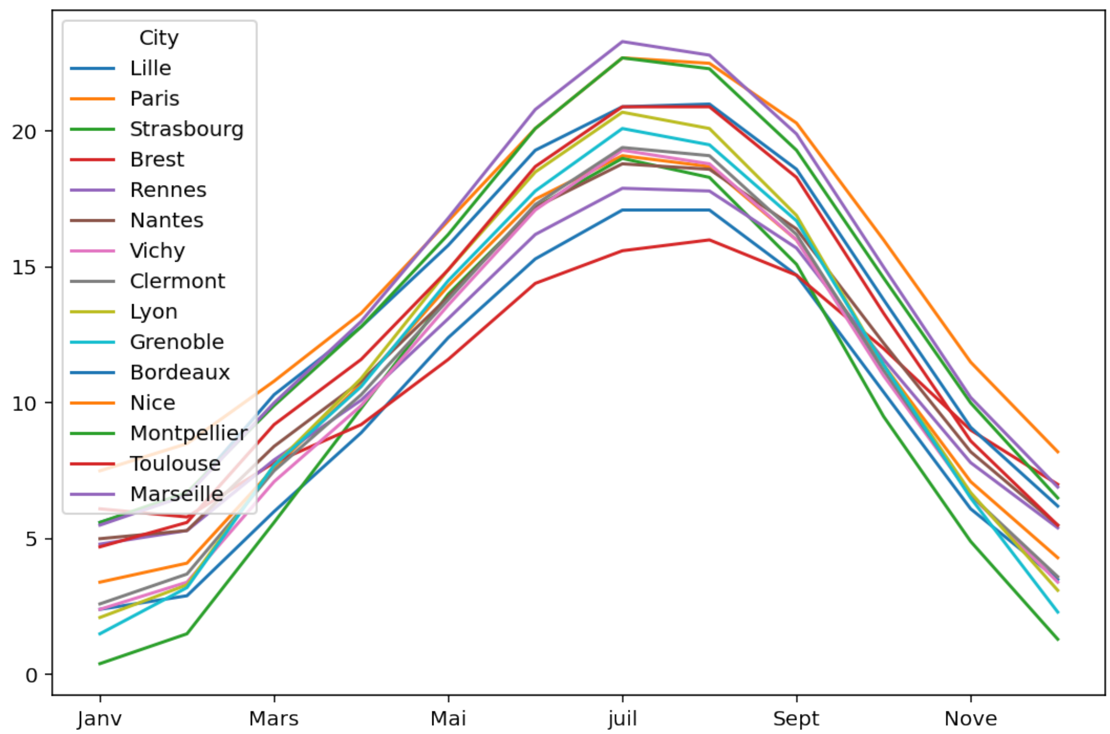
11.14 Transpose

The result is grouped in the wrong order since it sorts first the axis that was unstacked. We need to transpose the dataframe.

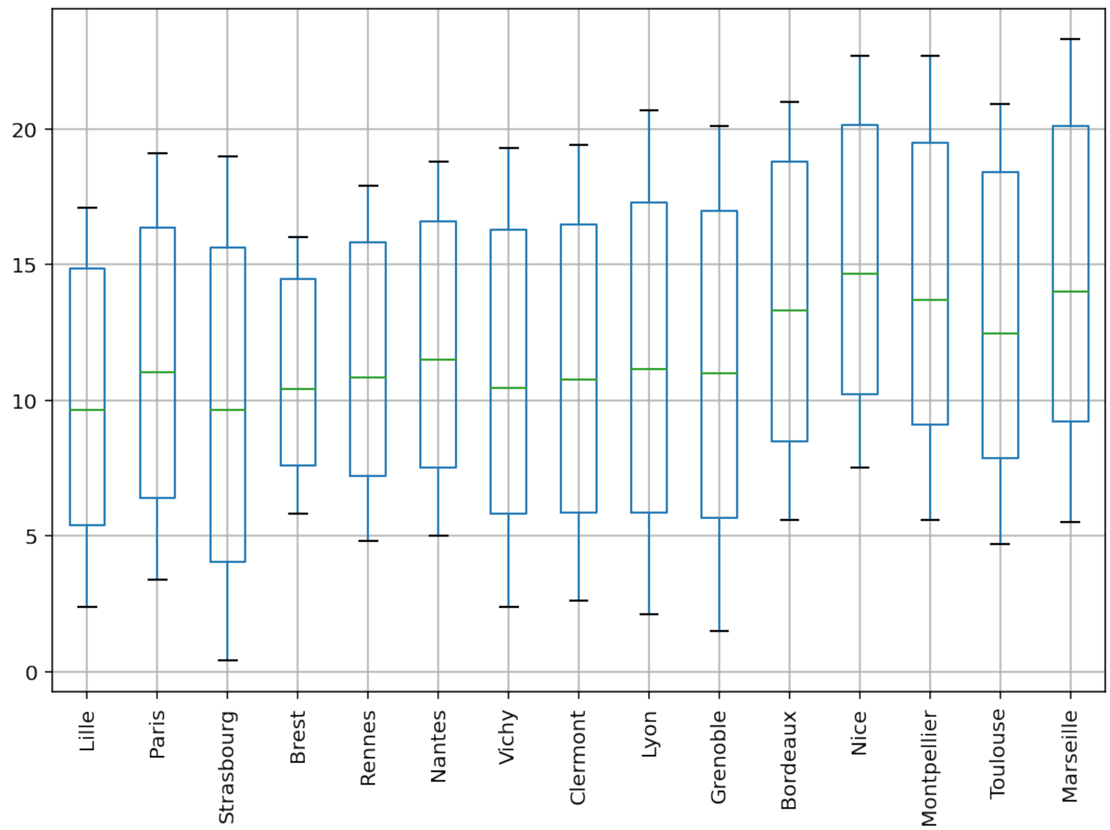
```
In [32]: city_temp = french_cities.iloc[:,:12].transpose()
         city_temp.plot()
```

```
/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/pandas/plotting/_matplotlib/core.py:1235
  ax.set_xticklabels(xticklabels)
```

```
Out[32]: <AxesSubplot:>
```



```
In [33]: city_temp.boxplot(rot=90);
```



11.15 Describing

```
In [34]: french_cities['Région'].describe()
```

```
Out[34]: count      15
         unique      4
         top        SE
         freq        7
         Name: Région, dtype: object
```

```
In [35]: french_cities['Région'].unique()
```

```
Out[35]: array(['NE', 'NO', 'SE', 'SO'], dtype=object)
```

```
In [36]: french_cities['Région'].value_counts()
```

```
Out[36]: SE    7
         NE    3
         NO    3
         SO    2
         Name: Région, dtype: int64
```

```
In [37]: # To save memory, we can convert it to a categorical column:
         french_cities["Région"] = french_cities["Région"].astype("category")
```

```
In [38]: french_cities.memory_usage()
```

```
Out[38]: Index     760
         Janv      120
         Févr      120
         Mars      120
         Avri      120
         Mai       120
         Juin      120
         juil      120
         Août      120
         Sept      120
         Octo      120
         Nove      120
         Déce      120
         Lati      120
         Long      120
         Mean      120
         Ampl      120
         Région    207
         dtype: int64
```

11.16 Data Aggregation/summarization

11.17 groupby

```
In [39]: fc_grouped_region = french_cities.groupby("Région")
         type(fc_grouped_region)
```

```
Out[39]: pandas.core.groupby.generic.DataFrameGroupBy
```

```
In [40]: for group_name, subdf in fc_grouped_region:
             print(group_name)
             print(subdf)
             print("")
```

```
NE
            Janv  Févr  Mars  Avri   Mai  Juin  juil  Août  Sept  Octo  Nove  \
City
Lille        2.4   2.9   6.0   8.9  12.4  15.3  17.1  17.1  14.7  10.4   6.1
Paris        3.4   4.1   7.6  10.7  14.3  17.5  19.1  18.7  16.0  11.4   7.1
Strasbourg   0.4   1.5   5.6   9.8  14.0  17.2  19.0  18.3  15.1   9.5   4.9

            Déce   Lati  Long   Mean  Ampl Région
City
Lille        3.5  50.38  3.04   9.73  14.7     NE
Paris        4.3  48.52  2.20  11.18  15.7     NE
Strasbourg   1.3  48.35  7.45   9.72  18.6     NE

NO
        Janv  Févr  Mars  Avri   Mai  Juin  juil  Août  Sept  Octo  Nove  \
City
Brest    6.1   5.8   7.8   9.2  11.6  14.4  15.6  16.0  14.7  12.0   9.0
Rennes   4.8   5.3   7.9  10.1  13.1  16.2  17.9  17.8  15.7  11.6   7.8
Nantes   5.0   5.3   8.4  10.8  13.9  17.2  18.8  18.6  16.4  12.2   8.2
```

```
        Déce   Lati  Long   Mean  Ampl Région
City
Brest    7.0  48.24 -4.29  10.77  10.2     NO
Rennes   5.4  48.05 -1.41  11.13  13.1     NO
Nantes   5.5  47.13 -1.33  11.69  13.8     NO

SE
             Janv  Févr  Mars  Avri   Mai  Juin  juil  Août  Sept  Octo  Nove  \
City
Vichy         2.4   3.4   7.1   9.9  13.6  17.1  19.3  18.8  16.0  11.0   6.6
Clermont      2.6   3.7   7.5  10.3  13.8  17.3  19.4  19.1  16.2  11.2   6.6
Lyon          2.1   3.3   7.7  10.9  14.9  18.5  20.7  20.1  16.9  11.4   6.7
Grenoble      1.5   3.2   7.7  10.6  14.5  17.8  20.1  19.5  16.7  11.4   6.5
Nice          7.5   8.5  10.8  13.3  16.7  20.1  22.7  22.5  20.3  16.0  11.5
Montpellier   5.6   6.7   9.9  12.8  16.2  20.1  22.7  22.3  19.3  14.6  10.0
Marseille     5.5   6.6  10.0  13.0  16.8  20.8  23.3  22.8  19.9  15.0  10.2

             Déce   Lati  Long   Mean  Ampl Région
City
Vichy         3.4  46.08  3.26  10.72  16.9     SE
Clermont      3.6  45.47  3.05  10.94  16.8     SE
Lyon          3.1  45.45  4.51  11.36  18.6     SE
Grenoble      2.3  45.10  5.43  10.98  18.6     SE
Nice          8.2  43.42  7.15  14.84  15.2     SE
Montpellier   6.5  43.36  3.53  13.89  17.1     SE
Marseille     6.9  43.18  5.24  14.23  17.8     SE

SO
          Janv  Févr  Mars  Avri   Mai  Juin  juil  Août  Sept  Octo  Nove  \
City
Bordeaux   5.6   6.6  10.3  12.8  15.8  19.3  20.9  21.0  18.6  13.8   9.1
Toulouse   4.7   5.6   9.2  11.6  14.9  18.7  20.9  20.9  18.3  13.3   8.6

          Déce   Lati  Long   Mean  Ampl Région
City
Bordeaux   6.2  44.50 -0.34  13.33  15.4     SO
Toulouse   5.5  43.36  1.26  12.68  16.2     SO
```

Chapter 12

PySpark

- Apache Spark was first released in 2014.
- It was originally developed by Matei Zaharia as a class project, and later a PhD dissertation, at University of California, Berkeley.
- Spark is written in Scala.
- All images come from Databricks.

- Apache Spark is a fast and general-purpose cluster computing system.
- It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs.
- Spark can manage "big data" collections with a small set of high-level primitives like `map`, `filter`, `groupby`, and `join`. With these common patterns we can often handle computations that are more complex than map, but are still structured.
- It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming.

12.1 Resilient distributed datasets

- The fundamental abstraction of Apache Spark is a read-only, parallel, distributed, fault-tolerent collection called a resilient distributed datasets (RDD).
- RDDs behave a bit like Python collections (e.g. lists).
- When working with Apache Spark we iteratively apply functions to every item of these collections in parallel to produce *new* RDDs.
- The data is distributed across nodes in a cluster of computers.

- Functions implemented in Spark can work in parallel across elements of the collection.
- The Spark framework allocates data and processing to different nodes, without any intervention from the programmer.
- RDDs automatically rebuilt on machine failure.

12.2 Lifecycle of a Spark Program

1. Create some input RDDs from external data or parallelize a collection in your driver program.
2. Lazily transform them to define new RDDs using transformations like `filter()` or `map()`
3. Ask Spark to cache() any intermediate RDDs that will need to be reused.
4. Launch actions such as count() and collect() to kick off a parallel computation, which is then optimized and executed by Spark.

12.3 Operations on Distributed Data

- Two types of operations: **transformations** and **actions**
- Transformations are *lazy* (not computed immediately)
- Transformations are executed when an action is run

12.4 Transformations (lazy)

```
map() flatMap()
filter()
mapPartitions() mapPartitionsWithIndex()
sample()
union() intersection() distinct()
groupBy() groupByKey()
reduceBy() reduceByKey()
sortBy() sortByKey()
join()
cogroup()
cartesian()
pipe()
coalesce()
repartition()
partitionBy()
...
```

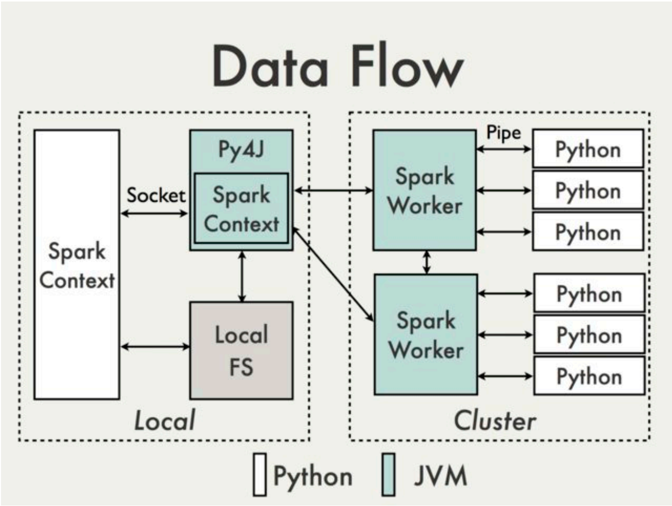
12.5 Actions

```
reduce()
collect()
count()
first()
take()
takeSample()
saveToCassandra()
takeOrdered()
saveAsTextFile()
saveAsSequenceFile()
saveAsObjectFile()
countByKey()
```

foreach()

12.6 Python API

PySpark uses Py4J that enables Python programs to dynamically access Java objects.



12.7 The SparkContext class

- When working with Apache Spark we invoke methods on an object which is an instance of the `pyspark.SparkContext` context.
- Typically, an instance of this object will be created automatically for you and assigned to the variable `sc`.
- The `parallelize` method in `SparkContext` can be used to turn any ordinary Python collection into an RDD;
  - normally we would create an RDD from a large file or an HBase table.

12.8 First example

PySpark isn't on sys.path by default, but that doesn't mean it can't be used as a regular library. You can address this by either symlinking pyspark into your site-packages, or adding pyspark to sys.path at runtime. findspark does the latter.

We have a spark context sc to use with a tiny local spark cluster with 4 nodes (will work just fine on a multicore machine).

If you use the workstation in room A111 run the code below before:

```
import findspark
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64",
os.environ["SPARK_HOME"] = "/export/spark-2.3.1-bin-hadoop2.7"

findspark.init()
```

```
In [1]: import sys
        sys.executable
```

```
Out[1]: '/usr/share/miniconda3/envs/big-data/bin/python'
```

```
In [2]: import sys, os
        os.environ["PYSPARK_PYTHON"] = sys.executable
```

```
In [3]: import pyspark
```

```
In [4]: sc = pyspark.SparkContext(master="local[*]", appName="FirstExample")
        sc.setLogLevel("ERROR")
```

```
In [5]: print(sc) # it is like a Pool Processor executor
```

```
<SparkContext master=local[*] appName=FirstExample>
```

12.9 Create your first RDD

```
In [6]: rdd = sc.parallelize(list(range(8))) # create collection
```

```
In [7]: rdd
```

```
Out[7]: ParallelCollectionRDD[0] at readRDDFromFile at PythonRDD.scala:262
```

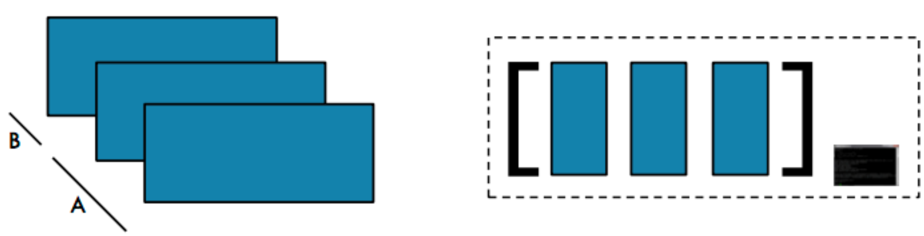
12.9.1 Exercise

Create a file `sample.txt`with lorem package. Read and load it into a RDD with the `textFile` spark function.

```
In [8]: import lorem

        with open("sample.txt","w") as f:
            f.write(lorem.text())

        rdd = sc.textFile("sample.txt")
```

12.9.2 Collect

Action / To Driver: Return all items in the RDD to the driver in a single list



Source: https://i.imgur.com/DUO6ygB.png

12.9.3 Exercise

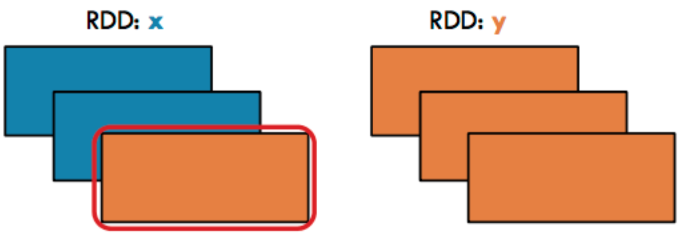
Collect the text you read before from the `sample.txt`file.

```
In [9]: rdd.collect()
```

```
Out[9]: ['Porro adipisci sit est sed dolor dolore etincidunt. Modi quisquam amet amet etincidunt dolore
          '',
          'Velit quaerat sed consectetur amet. Quisquam modi consectetur aliquam neque est dolor. Neque e
          '',
          'Voluptatem neque etincidunt labore non. Ut consectetur sed etincidunt voluptatem amet etincidu
          '',
          'Eius amet quiquia quisquam etincidunt. Quisquam aliquam labore dolor velit magnam voluptatem.
          '',
          'Neque quaerat modi quaerat numquam. Dolorem ut porro tempora amet. Numquam quiquia consectetur
          '',
          'Modi non voluptatem magnam dolor. Sit numquam sit modi consectetur quisquam. Ut dolore modi mo
```

12.9.4 Map

Transformation / Narrow: Return a new RDD by applying a function to each element of this RDD



Source: http://i.imgur.com/PxNJf0U.png

```
In [10]: rdd = sc.parallelize(list(range(8)))
         rdd.map(lambda x: x ** 2).collect() # Square each element
```

```
Out[10]: [0, 1, 4, 9, 16, 25, 36, 49]
```

12.9.5 Exercise

Replace the lambda function by a function that contains a pause (sleep(1)) and check if the `map` operation is parallelized.

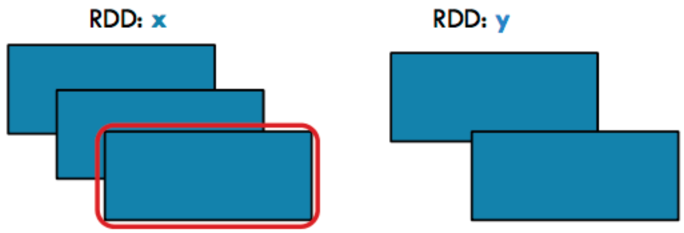
```
In [11]: from time import sleep
         def square(x):
             sleep(1)
             return x**2

         %time rdd.map(square).collect()
```

```
CPU times: user 0 ns, sys: 6.51 ms, total: 6.51 ms
Wall time: 4.06 s
```

```
Out[11]: [0, 1, 4, 9, 16, 25, 36, 49]
```

12.9.6 Filter

Transformation / Narrow: Return a new RDD containing only the elements that satisfy a predicate



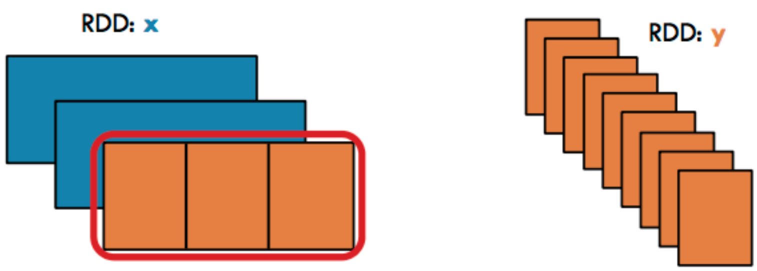
Source: http://i.imgur.com/GFyji4U.png

```
In [12]: # Select only the even elements
         rdd.filter(lambda x: x % 2 == 0).collect()
```

```
Out[12]: [0, 2, 4, 6]
```

12.9.7 FlatMap

Transformation / Narrow: Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results




```
In [13]: rdd = sc.parallelize([1,2,3])
         rdd.flatMap(lambda x: (x, x*100, 42)).collect()
```

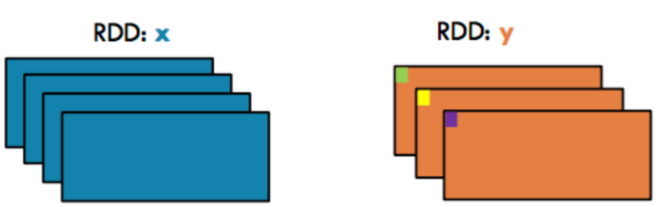
```
Out[13]: [1, 100, 42, 2, 200, 42, 3, 300, 42]
```

12.9.8 Exercise

Use FlatMap to clean the text from `sample.txt`file. Lower, remove dots and split into words.

12.9.9 GroupBy

Transformation / Wide: Group the data in the original RDD. Create pairs where the key is the output of a user function, and the value is all items for which the function yields this key.

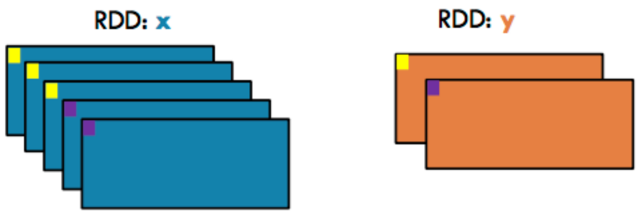



```
In [14]: rdd = sc.parallelize(['John', 'Fred', 'Anna', 'James'])
         rdd = rdd.groupBy(lambda w: w[0])
         [(k, list(v)) for (k, v) in rdd.collect()]
```

```
Out[14]: [('J', ['John', 'James']), ('F', ['Fred']), ('A', ['Anna'])]
```

12.9.10 GroupByKey

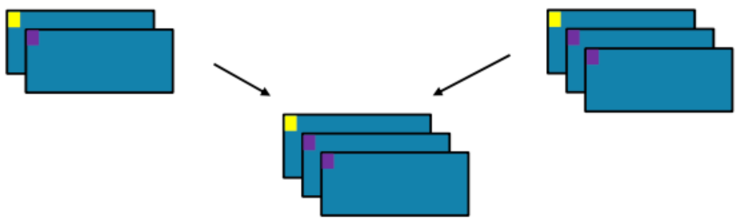
Transformation / Wide: Group the values for each key in the original RDD. Create a new pair where the original key corresponds to this collected group of values.




```
In [15]: rdd = sc.parallelize([('B',5),('B',4),('A',3),('A',2),('A',1)])
         rdd = rdd.groupByKey()
         [(j[0], list(j[1])) for j in rdd.collect()]

Out[15]: [('B', [5, 4]), ('A', [3, 2, 1])]
```

12.9.11 Join

Transformation / Wide: Return a new RDD containing all pairs of elements having the same key in the original RDDs



```
In [16]: x = sc.parallelize([("a", 1), ("b", 2)])
         y = sc.parallelize([("a", 3), ("a", 4), ("b", 5)])
         x.join(y).collect()

Out[16]: [('b', (2, 5)), ('a', (1, 3)), ('a', (1, 4))]
```

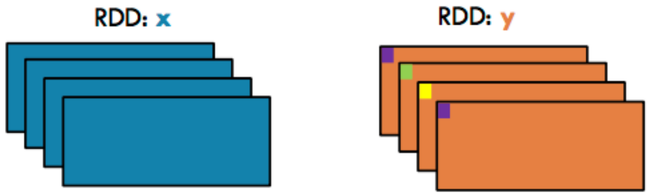
12.9.12 Distinct

Transformation / Wide: Return a new RDD containing distinct items from the original RDD (omitting all duplicates)

```
In [17]: rdd = sc.parallelize([1,2,3,3,4])
         rdd.distinct().collect()

Out[17]: [2, 4, 1, 3]
```

12.9.13 KeyBy

Transformation / Narrow: Create a Pair RDD, forming one pair for each item in the original RDD. The pair's key is calculated from the value via a user-supplied function.




```
In [18]: rdd = sc.parallelize(['John', 'Fred', 'Anna', 'James'])
         rdd.keyBy(lambda w: w[0]).collect()
```

```
Out[18]: [('J', 'John'), ('F', 'Fred'), ('A', 'Anna'), ('J', 'James')]
```

12.10 Actions

12.10.1 Map-Reduce operation

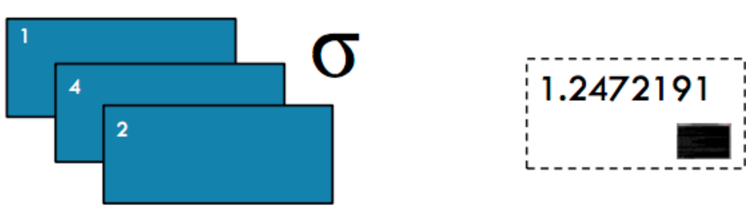
Action / To Driver: Aggregate all the elements of the RDD by applying a user function pairwise to elements and partial results, and return a result to the driver


```
In [19]: from operator import add
         rdd = sc.parallelize(list(range(8)))
         rdd.map(lambda x: x ** 2).reduce(add) # reduce is an action!
```

```
Out[19]: 140
```

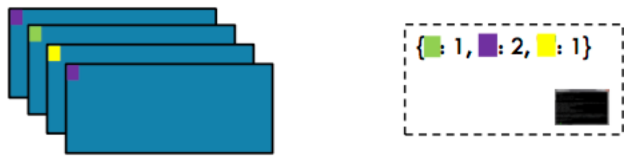
12.10.2 Max, Min, Sum, Mean, Variance, Stdev

Action / To Driver: Compute the respective function (maximum value, minimum value, sum, mean, variance, or standard deviation) from a numeric RDD



12.10.3 CountByKey

Action / To Driver: Return a map of keys and counts of their occurrences in the RDD




```
In [20]: rdd = sc.parallelize([('J', 'James'), ('F','Fred'),
                              ('A','Anna'), ('J','John')])

         rdd.countByKey()
```

```
Out[20]: defaultdict(int, {'J': 2, 'F': 1, 'A': 1})
```

```
In [21]: # Stop the local spark cluster
         sc.stop()
```

12.10.4 Exercise 10.1 Word-count in Apache Spark

- Write the sample text file

```
In [22]: from lorem import text
         with open('sample.txt','w') as f:
             f.write(text())
```

- Create the rdd with `SparkContext.textFile method`
- lower, remove dots and split using `rdd.flatMap`
- use `rdd.map` to create the list of key/value pair (word, 1)
- `rdd.reduceByKey` to get all occurences
- `rdd.takeOrdered`to get sorted frequencies of words

All documentation is available here for textFile and here for RDD.
For a global overview see the Transformations section of the programming guide

```
In [23]: import pyspark

         sc = pyspark.SparkContext(master="local[*]", appName="wordcount")
         sc.setLogLevel("ERROR")
```

```
In [24]: rdd = sc.textFile("sample.txt")
```

```
In [25]: (rdd.flatMap(lambda line: line.lower().replace("."," ").split())
             .map(lambda w : (w,1))
             .reduceByKey(lambda w, c: w + c)
             .sortBy( lambda w : -w[1]).collect())
```

```
Out[25]: [('quisquam', 17),
          ('labore', 15),
          ('ut', 14),
          ('dolor', 14),
          ('porro', 13),
          ('tempora', 13),
          ('aliquam', 13),
          ('dolorem', 13),
          ('consectetur', 12),
          ('adipisci', 11),
          ('dolore', 11),
          ('est', 10),
```

```
('neque', 10),
('voluptatem', 10),
('non', 10),
('magnam', 9),
('sed', 9),
('etincidunt', 9),
('sit', 9),
('ipsum', 8),
('velit', 8),
('quiquia', 7),
('quaerat', 7),
('modi', 7),
('numquam', 7),
('eius', 5),
('amet', 4)]
```

```
In [26]: sc.stop()
```

12.11 SparkSession

Since SPARK 2.0.0, SparkSession provides a single point of entry to interact with Spark functionality and allows programming Spark with DataFrame and Dataset APIs.

12.11.1 π computation example

- We can estimate an approximate value for π using the following Monte-Carlo method:

1. Inscribe a circle in a square
2. Randomly generate points in the square
3. Determine the number of points in the square that are also in the circle
4. Let r be the number of points in the circle divided by the number of points in the square, then $\pi \approx 4r$.

- Note that the more points generated, the better the approximation

See this tutorial.

```
In [27]: import sys
         from random import random
         from operator import add

         from pyspark.sql import SparkSession

         spark = (SparkSession.builder.master("local[*]")
                  .appName("PythonPi")
                  .getOrCreate())

         partitions = 8
         n = 100000 * partitions

         def f(_):
             x = random() * 2 - 1
             y = random() * 2 - 1
             return 1 if x ** 2 + y ** 2 <= 1 else 0

         count = spark.sparkContext.parallelize(range(1, n+1), partitions).map(f).reduce(add)
         print("Pi is roughly %f" % (4.0 * count / n))

         spark.stop()
```

```
Pi is roughly 3.142260
```

12.11.2 Exercise 9.2

Using the same method than the PI computation example, compute the integral

$$I = \int_0^1 \exp(-x^2)dx$$

You can check your result with numpy

```
In [28]: # numpy evaluates solution using numeric computation.
         # It uses discrete values of the function
         import numpy as np
         x = np.linspace(0,1,1000)
         np.trapz(np.exp(-x*x),x)
```

```
Out[28]: 0.7468240713763741
```

```
In [29]: # numpy and scipy evaluates solution using numeric computation. It uses discrete values
         # of the function
         import numpy as np
         from scipy.integrate import quad
         quad(lambda x: np.exp(-x*x), 0, 1)
         # note: the solution returned is complex
```

```
Out[29]: (0.7468241328124271, 8.291413475940725e-15)
```

12.11.3 Correlation between daily stock

- Data preparation

```
In [30]: import os  # library to get directory and file paths
         import tarfile # this module makes possible to read and write tar archives

         def extract_data(name, where):
             datadir = os.path.join(where,name)
             if not os.path.exists(datadir):
                print("Extracting data...")
                tar_path = os.path.join(where, name+'.tgz')
                with tarfile.open(tar_path, mode='r:gz') as data:
                   data.extractall(where)

         extract_data('daily-stock','data') # this function call will extract json files
```

```
In [31]: import json
         import pandas as pd
         import os, glob

         here = os.getcwd()
         datadir = os.path.join(here,'data','daily-stock')
         filenames = sorted(glob.glob(os.path.join(datadir, '*.json')))
         filenames
```

```
Out[31]: ['/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aet.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/afl.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aig.json',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/al.json',
```

```
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/amgn.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/avy.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/b.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/bwa.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ge.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hal.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hp.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hpq.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ibm.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jbl.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jpm.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/luv.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/met.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/pcg.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/tgt.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/usb.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/xom.json']
```

In [32]:
```
%rm data/daily-stock/*.h5
```

In [33]:
```
from glob import glob
import os, json
import pandas as pd

for fn in filenames:
    with open(fn) as f:
        data = [json.loads(line) for line in f]

    df = pd.DataFrame(data)

    out_filename = fn[:-5] + '.h5'
    df.to_hdf(out_filename, '/data')
    print("Finished : %s" % out_filename.split(os.path.sep)[-1])

filenames = sorted(glob(os.path.join('data', 'daily-stock', '*.h5')))  # data/json/*.json
```

```
Finished : aet.h5
Finished : afl.h5
Finished : aig.h5
Finished : al.h5
Finished : amgn.h5
Finished : avy.h5
Finished : b.h5
Finished : bwa.h5
Finished : ge.h5
Finished : hal.h5
Finished : hp.h5
Finished : hpq.h5
Finished : ibm.h5
Finished : jbl.h5
Finished : jpm.h5
Finished : luv.h5
Finished : met.h5
Finished : pcg.h5
Finished : tgt.h5
Finished : usb.h5
```

```
Finished : xom.h5
```

12.11.4 Sequential code

```
In [34]: filenames

Out[34]: ['data/daily-stock/aet.h5',
          'data/daily-stock/afl.h5',
          'data/daily-stock/aig.h5',
          'data/daily-stock/al.h5',
          'data/daily-stock/amgn.h5',
          'data/daily-stock/avy.h5',
          'data/daily-stock/b.h5',
          'data/daily-stock/bwa.h5',
          'data/daily-stock/ge.h5',
          'data/daily-stock/hal.h5',
          'data/daily-stock/hp.h5',
          'data/daily-stock/hpq.h5',
          'data/daily-stock/ibm.h5',
          'data/daily-stock/jbl.h5',
          'data/daily-stock/jpm.h5',
          'data/daily-stock/luv.h5',
          'data/daily-stock/met.h5',
          'data/daily-stock/pcg.h5',
          'data/daily-stock/tgt.h5',
          'data/daily-stock/usb.h5',
          'data/daily-stock/xom.h5']
```

```
In [35]: with pd.HDFStore('data/daily-stock/aet.h5') as hdf:
             # This prints a list of all group names:
             print(hdf.keys())
```

```
['/data']
```

```
In [36]: df_test = pd.read_hdf('data/daily-stock/aet.h5')
```

```
In [37]: %%time

         series = []
         for fn in filenames:   # Simple map over filenames
             series.append(pd.read_hdf(fn)["close"])

         results = []

         for a in series:    # Doubly nested loop over the same collection
             for b in series:
                 if not (a == b).all():     # Filter out comparisons of the same series
                     results.append(a.corr(b))  # Apply function

         result = max(results)
         result
```

```
CPU times: user 1.3 s, sys: 63.1 ms, total: 1.36 s
Wall time: 1.35 s
```

```
Out[37]: 0.9413176064560879
```

12.11.5 Exercise 9.3

Parallelize the code above with Apache Spark.

- Change the filenames because of the Hadoop environment.

```
In [38]: import os, glob

         here = os.getcwd()
         filenames = sorted(glob.glob(os.path.join(here,'data', 'daily-stock', '*.h5')))
         filenames
```

```
Out[38]: ['/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aet.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/afl.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aig.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/al.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/amgn.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/avy.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/b.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/bwa.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ge.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hal.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hp.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hpq.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ibm.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jbl.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jpm.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/luv.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/met.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/pcg.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/tgt.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/usb.h5',
          '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/xom.h5']
```

If it is not started don't forget the PySpark context

```
In [39]: ### Parallel code
         import pandas as pd

         sc = pyspark.SparkContext(master="local[*]", appName="series")
         sc.setLogLevel("ERROR")

         rdd = sc.parallelize(filenames)
         series = rdd.map(lambda fn: pd.read_hdf(fn)['close'])

         corr = (series.cartesian(series)
                       .filter(lambda ab: not (ab[0] == ab[1]).all())
                       .map(lambda ab: ab[0].corr(ab[1]))
                       .max())

         print(corr)
         sc.stop()
```

```
0.9413176064560879
```

Computation time is slower because there is a lot of setup, workers creation, there is a lot of communications the correlation function is too small

12.11.6 Exercise 9.4 Fasta file example

Use a RDD to calculate the GC content of fasta file nucleotide-sample.txt:

$$\frac{G+C}{A+T+G+C} \times 100\%$$

Create a rdd from fasta file genome.txt in data directory and count 'G' and 'C' then divide by the total number of bases.

```
In [40]: import pyspark
         sc = pyspark.SparkContext(master="local[*]", appName="series")
         sc.setLogLevel("ERROR")

         genome = sc.textFile('data/genome.txt')

In [41]: lines = genome.flatMap(lambda line: line.split())
         g = lines.map(lambda line: line.count("G")).sum()
         c = lines.map(lambda line: line.count("C")).sum()
         (g + c) / lines.map(lambda line:len(line)).sum()

Out[41]: 0.5048333333333334
```

12.11.7 Another example

Compute the most frequent sequence with 5 bases.

```
In [42]: def group_characters(line, n=5):
             result = ''
             i = 0
             for ch in line:
                 result = result + ch
                 i = i + 1
                 if (i % n) == 0:
                     yield result
                     result = ''

         def group_and_split(line):
             return [sequence for sequence in group_characters(line)]

         sequences = genome.flatMap(group_and_split)
         sequences.take(3)

Out[42]: ['GATCA', 'ATGAG', 'GTGGA']

In [43]: counts = sequences.map(lambda w: (w, 1)).reduceByKey(lambda x, y: x + y).sortBy(lambda v:-v[1])
         counts.take(10)

Out[43]: [('CTGTG', 59),
          ('CCCAG', 55),
          ('CCTGG', 52),
          ('AAAAA', 49),
          ('TGCTG', 42),
          ('TGTGT', 41),
          ('CCACC', 39),
          ('GGCTG', 38),
          ('CACCA', 37),
          ('GTGGG', 37)]

In [44]: sc.stop()
```

Chapter 13

Basic Commands in the Unix Shell

For windows 10 users, activate bash.

Or install Jupyter Lab

```
conda install jupyterlab -c conda-forge
jupyter lab
```

13.1 Unix Shell

The shell is a command programming language that provides an interface to the UNIX operating system. Documentation of unix command is displayed by command `man`. Exemple:

```
man whoami
```

```
In [1]: #%%bash
        #man whoami
```

13.2 Directories

The shell should start you in your home directory. This is your individual space on the UNIX system for your files. You can find out the name of your current working directory with the unix command `pwd`.

In the terminal, type the letters 'p', 'w', 'd', and then "enter" - always conclude each command by pressing the "enter" key. The response that follows on the next line will be the name of your home directory, where the name following the last slash should be your username.) The directory structure can be conceptualized as an inverted tree.

In the jupyter notebook, unix shell command can be executed using the escape character "!" or add `%%bash` to the cell first line. You can type command directly in a terminal without the "!".

```
In [2]: #%%bash
        #pwd
```

Some unix command (not all) are also jupyter magic command like %pwd

```
In [3]: #%pwd
```

13.3 Home directory

No matter where in the directory structure you are, you can always get back to your home directory with `cd`.

13.3.1 Create a new subdirectory named ”primer” :

```
mkdir primer
```

```
In [4]: #%%bash
        #rm -rf primer  # remove primer directory if it exists
        #mkdir  primer  # make the new directory
```

Now change to the ”primer” subdirectory, making it your current working directory:

```
cd primer
pwd
```

```
In [5]: #%cd primer
```

```
In [6]: #pwd
```

13.4 Files

Create a file using `date` command and `whoami`:

```
date >> first.txt
whoami >> first.txt
```

date and whoami are not jupyter magic commands

```
In [7]: #%%bash
        #
        #date >> first.txt
        #whoami >> first.txt
```

13.4.1 List files and directories

Files live within directories. You can see a list of the files in your ”primer” directory (which should be your current working directory) by typing:

```
ls
```

```
In [8]: #%%bash
        #ls
```

13.4.2 Display file content

You can view a text file with the following command:

```
cat first.txt
```

(”cat” is short for concatenate - you can use this to display multiple files together on the screen.) If you have a file that is longer than your 24-line console window, use instead ”more” to list one page at a time or ”less” to scroll the file down and up with the arrow keys. Don’t use these programs to try to display binary (non-text) files on your console - the attempt to print the non-printable control characters might alter your console settings and render the console unusable.

```
In [9]: #%%bash
        #cat first.txt
```

- Copy file ”first” using the following command:

```
cp first.txt 2nd.txt
```

By doing this you have created a new file named ”2nd.txt” which is a duplicate of file ”first.txt”. Geet he file listing with:

```
ls
```

```
In [10]: #%%bash
         #cp first.txt 2nd.txt
         #ls
```

- Now rename the file ”2nd” to ”second”:

  ```
  mv 2nd.txt second.txt
  ```

  Listing the files still shows two files because you haven’t created a new file, just changed an existing file’s name:

  ```
  ls
  ```

```
In [11]: #%%bash
         #mv 2nd.txt second.txt
         #ls
```

If you ”cat” the second file, you’ll see the same sentence as in your first file:

```
cat second.txt
```

```
In [12]: #%%bash
         #cat second.txt
```

”mv” will allow you to move files, not just rename them. Perform the following commands:

```
mkdir sub
mv second.txt sub
ls sub
ls
```

(where ”username” will be your username and ”group” will be your group name). Among other things, this lists the creation date and time, file access permissions, and file size in bytes. The letter ’d’ (the first character on the line) indicates the directory names.

```
In [13]: #%%bash
         #mkdir sub
         #mv second.txt sub
         #ls sub
```

This creates a new subdirectory named ”sub”, moves ”second” into ”sub”, then lists the contents of both directories. You can list even more information about files by using the ”-l” option with ”ls”:

```
In [14]: #%%bash
         #ls -l
```

Next perform the following commands:

```
cd sub
pwd
ls -l
cd ..
pwd
```

```
In [15]: #%%bash
         ## go to sub directory
         #cd sub
         ## current working directory
         #pwd
         ## list files with permissions
         #ls -l
         ## go to parent directory
         #cd ..
         ## current working directory
         #pwd
```

Finally, clean up the duplicate files by removing the "second.txt" file and the "sub" subdirectory:

```
rm sub/second.txt
rmdir sub
ls -l
cd
```

This shows that you can refer to a file in a different directory using the relative path name to the file (you can also use the absolute path name to the file - something like "/Users/username/primer/sub/second.txt", depending on your home directory). You can also include the ".." within the path name (for instance, you could have referred to the file as "../primer/sub/second.txt").

```
In [16]: #%%bash
         #rm -f sub/second.txt
         #rmdir sub
         #ls -l
         #cd ..
         #rm -rf primer
```

13.5 Connect to a server

Remote login to another machine can be accomplished using the "ssh" command:

```
ssh -l mylogin host
```

or

```
ssh mylogin@host
```

where "myname" will be your username on the remote system (possibly identical to your username on this system) and "host" is the name (or IP address) of the machine you are logging into.

Transfer files between machines using "scp". - To copy file "myfile" from the remote machine named "host":

```
scp myname@host:myfile .
```

- To copy file "myfile" from the local machine to the remote named "host":

  ```
  scp myfile myname@host:
  ```

- Use `ssh -r` option to copy a directory (The "." refers to your current working directory, meaning that the destination for "myfile" is your current directory.)

13.5.1 Exercise

- Copy a file to the server `svmass2.mass.uhb.fr`
- Log on to this server and display this file with `cat`

13.6 Secure copy (scp)

Synchronize big-data directory on the cluster:

```
scp -r big-data svmass2:
```

This a secure copy of big-data directory to the server.
or

```
rsync -e ssh -avrz big-data svmass2:
```

It synchronizes the local directory big-data with the remote repository big-data on svmass2 server

13.7 Summary Of Basic Shell Commands

```
% pico myfile                # text edit file "myfile"
% ls                         # list files in current directory
% ls -l                      # long format listing
% touch myfile               # create new empty file "myfile"
% cat myfile                 # view contents of text file "myfile"
% more myfile                # paged viewing of text file "myfile"
% less myfile                # scroll through text file "myfile"
% head myfile                # view 10 first lines of text file "myfile"
% tail myfile                # view 10 last lines of text file "myfile"
% cp srcfile destfile        # copy file "srcfile" to new file "destfile"
% mv oldname newname         # rename (or move) file "oldname" to "newname"
% rm myfile                  # remove file "myfile"
% mkdir subdir               # make new directory "subdir"
% cd subdir                  # change current working directory to "subdir"
% rmdir subdir               # remove (empty) directory "subdir"
% pwd                        # display current working directory
% date                       # display current date and time of day
% ssh -l myname host         # remote shell login of username "myname" to "host"
% scp myname@host:myfile .   # remote copy of file "myfile" to current directory
% scp myfile myname@host:    # copy of file "myfile" to remote server
% firefox &                  # start Firefox web browser (in background)
% jobs                       # display programs running in background
% kill %n                    # kill job number n (use jobs to get this number)
% man -k "topic"             # search manual pages for "topic"
% man command                # display man page for "command"
% exit                       # exit a terminal window
% logout                     # logout of a console session
```

13.8 Redirecting

Redirection is usually implemented by placing characters <,>,|,>> between commands.

- Use > to redirect output.

  ```
  ls *.ipynb > file_list.txt
  ```

  executes `ls`, placing the output in file_list.txt, as opposed to displaying it at the terminal, which is the usual destination for standard output. This will clobber any existing data in file1.

```
In [17]: #%%bash
         #ls *.ipynb > file_list.txt
```

- Use < to redirect input.

  ```
  wc < file_list.txt
  ```

  executes `wc`, with file_list.txt as the source of input, as opposed to the keyboard, which is the usual source for standard input.

13.8.1 Python example

```
In [18]: %%file test_stdin.py
         #!/usr/bin env python
         import sys

         # input comes from standard input
         k = 0
         for file in sys.stdin:
             k +=1
             print('file {} : {}'.format(k,file))
```

```
Overwriting test_stdin.py
```

```
In [19]: # %%bash
         # python test_stdin.py < file_list.txt
```

You can combine the two capabilities: read from an input file and write to an output file.

```
In [20]: # %%bash
         # python test_stdin.py < file_list.txt > output.txt
```

```
In [21]: # %%bash
         # cat output.txt
```

To append output to the end of the file, rather than clobbering it, the >> operator is used:
date >> output.txt
It will append the today date to the end of the file output.txt

```
In [22]: # %%bash
         # date >> output.txt
         # cat output.txt
```

13.9 Permissions

Every file on the system has associated with it a set of permissions. Permissions tell UNIX what can be done with that file and by whom. There are three things you can (or can't) do with a given file: - read, - write (modify), - execute.

Unix permissions specify what can 'owner', 'group' and 'all' can do.

If you try ls -l on the command prompt you get something like the following:

```
-rw-r--r--  1 navaro  staff   15799  5 oct 15:57 01.MapReduce.ipynb
-rw-r--r--  1 navaro  staff   18209 12 oct 16:04 02.Containers.ipynb
-rw-r--r--  1 navaro  staff   37963 12 oct 21:28 03.ParallelComputation.ipynb
```

Three bits specify access permissions: - **r** read, - **w** access, - **w** execute.

13.9.1 Example

`rwxr-xr--`

- the owner can do anything with the file,
- group owners and the can only read or execute it.
- rest of the world can only read

13.10 chmod

To set/modify a file's permissions you need to use the chmod program. Of course, only the owner of a file may use chmod to alter a file's permissions. chmod has the following syntax:

```
chmod [options] mode file(s)
```

- The 'mode' part specifies the new permissions for the file(s) that follow as arguments. A mode specifies which user's permissions should be changed, and afterwards which access types should be changed.
- We use `+` or `-` to change the mode for owner, group and the rest of the world.
- The permissions start with a letter specifying what users should be affected by the change.

Original permissions of script.py are `rw-------`

- `chmod u+x script.py` set permissions to `rwx------`
- `chmod a+x script.py` set permissions to `rwx--x--x`
- `chmod g+r script.py` set permissions to `rwxr-x--x`
- `chmod o-x script.py` set permissions to `rwxr-x---`
- `chmod og+w script.py` set permissions to `rwxrwx-w-`

13.11 Pipelining

```
ls | grep ipynb
```

executes `ls`, using its output as the input for `grep`.

13.11.1 Exercice 11.1

- Pipe `cat *.ipynb` output to `sort` command.
- Pipe `ls` output to `wc` command.
- Pipe `cat 11.UnixCommands.ipynb` to `less` command.

13.12 Chained pipelines

The redirection and piping tokens can be chained together to create complex commands.

13.12.1 Exercice 11.2

Use unix commands chained to display word count of file `sample.txt`.

Hints:

- `fmt -n` takes text as input and reformats it into paragraphs with no line longer than n.
- `sort` sort the output alphabetically
- `tr -d str` delete the string str from the output
- `uniq -c` writes a copy of each unique input and precede each word with the count of the number of occurences.

```
In [23]: from lorem import text
         with open('sample.txt', 'w') as f:
             f.write(text())
```

13.12.2 Exercice 11.3

- Create a python script mapper.py to count words from stdin. The script prints out every word found in stdin with the value 1 separate by a tab.

```
Consectetur 1
adipisci    1
quiquia 1
sit 1
```

File `mapper.py` must be executable.

```
In [24]: # %%bash
         # chmod +x mapper.py
```

13.12.3 Exercice 11.4

- Create a python script reducer.py to read output from mapper.py. The script prints out every word and number of occurences.

```
cat sample.txt | ./mapper.py | ./reducer.py
```

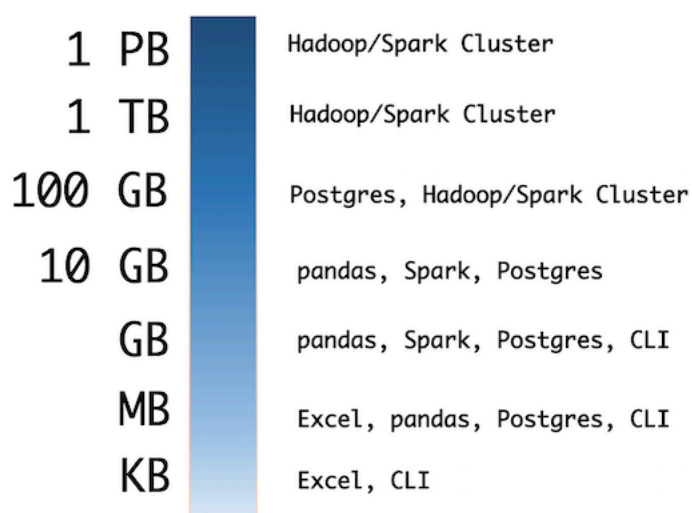
```
7   porro
7   eius
6   non
6   dolore
```

```
In [25]: # %%bash
         # chmod +x ./reducer.py
```

Chapter 14

Hadoop

- Data sets that are so large or complex that traditional data processing application software is inadequate to deal with them.
- Data analysis requires massively parallel software running on several servers.
- **Volume, Variety, Velocity, Variability and Veracity** describe Big Data properties.



- Framework for running applications on large cluster.
- The Hadoop framework transparently provides applications both reliability and data motion.
- Hadoop implements the computational paradigm named **Map/Reduce**, where the application is divided into many small fragments of work, each of which may be executed or re-executed on any node in the cluster.
- It provides a distributed file system (HDFS) that stores data on the compute nodes, providing very high aggregate bandwidth across the cluster.
- Both MapReduce and the **Hadoop Distributed File System** are designed so that node failures are automatically handled by the framework.

14.1 HDFS

- It is a distributed file systems.
- HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware.
- HDFS is suitable for applications that have large data sets.
- HDFS provides interfaces to move applications closer to where the data is located. The computation is much more efficient when the size of the data set is huge.
- HDFS consists of a single NameNode with a number of DataNodes which manage storage.
- HDFS exposes a file system namespace and allows user data to be stored in files.
  1. A file is split by the NameNode into blocks stored in DataNodes.
  2. The **NameNode** executes operations like opening, closing, and renaming files and directories.
  3. The **Secondary NameNode** stores information from **NameNode**.
  4. The **DataNodes** manage perform block creation, deletion, and replication upon instruction from the NameNode.
  5. The placement of replicas is optimized for data reliability, availability, and network bandwidth utilization.
  6. User data never flows through the NameNode.
- Files in HDFS are write-once and have strictly one writer at any time.
- The DataNode has no knowledge about HDFS files.

14.2 Accessibility

All HDFS commands are invoked by the bin/hdfs Java script:

```
hdfs [SHELL_OPTIONS] COMMAND [GENERIC_OPTIONS] [COMMAND_OPTIONS]
```

14.3 Manage files and directories

```
hdfs dfs -ls -h -R # Recursively list subdirectories with human-readable file sizes.
hdfs dfs -cp  # Copy files from source to destination
hdfs dfs -mv  # Move files from source to destination
hdfs dfs -mkdir /foodir # Create a directory named /foodir
hdfs dfs -rmr /foodir   # Remove a directory named /foodir
hdfs dfs -cat /foodir/myfile.txt #View the contents of a file named /foodir/myfile.txt
```

14.4 Transfer between nodes

14.4.1 put

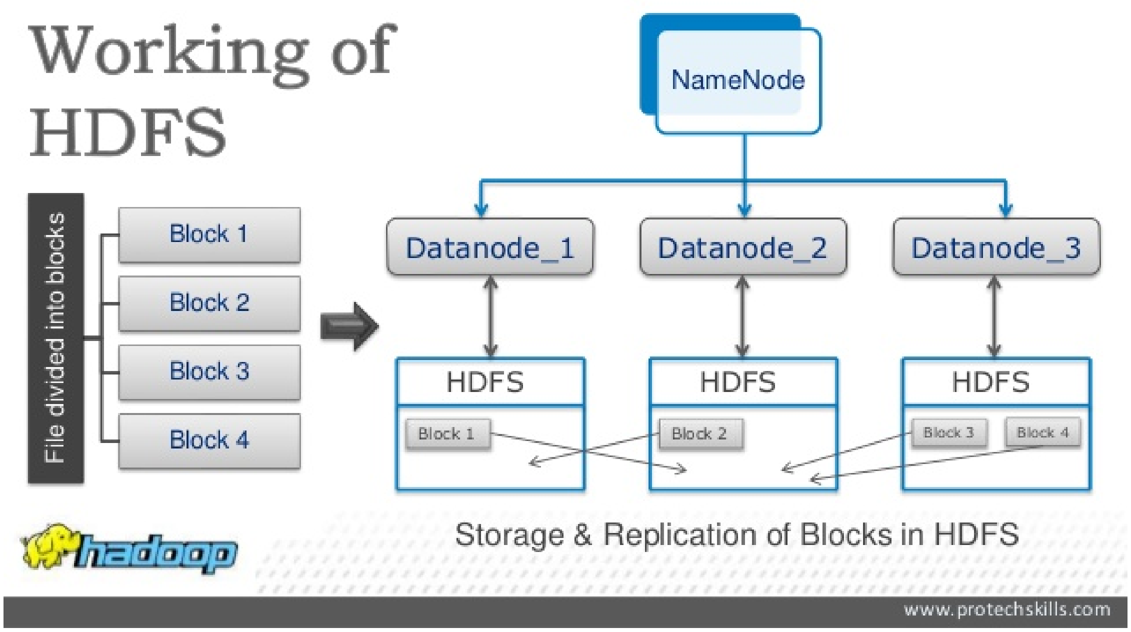
```
hdfs fs -put [-f] [-p] [-l] [-d] [ - | <localsrc1> .. ]. <dst>
```

Copy single src, or multiple srcs from local file system to the destination file system.
Options:

```
-p : Preserves rights and modification times.
-f : Overwrites the destination if it already exists.
```

```
hdfs fs -put localfile /user/hadoop/hadoopfile
hdfs fs -put -f localfile1 localfile2 /user/hadoop/hadoopdir
```

Similar to the fs -put command - `moveFromLocal` : to delete the source localsrc after copy. - `copyFromLocal` : source is restricted to a local file - `copyToLocal` : destination is restricted to a local file



The Name Node is not in the data path. The Name Node only provides the map of where data is and where data should go in the cluster (file system metadata).

14.5 Hadoop cluster

- 8 computers: sve1 -> sve9

14.5.1 NameNode Web Interface (HDFS layer)

http://svmass2.mass.uhb.fr:50070

The name node web UI shows you a cluster summary including information about total/remaining capacity, live and dead nodes. Additionally, it allows you to browse the HDFS namespace and view the contents of its files in the web browser. It also gives access to the local machine's Hadoop log files.

14.5.2 Secondary Namenode Information.

http://svmass2.mass.uhb.fr:50090/

14.5.3 Datanode Information.

- http://svpe1.mass.uhb.fr:50075/
- http://svpe2.mass.uhb.fr:50075/
- ...
- http://svpe8.mass.uhb.fr:50075/
- http://svpe9.mass.uhb.fr:50075/

To do following hands on you can switch to JupyterLab.
Just go to this following address http://localhost:9000/lab

- Check that your HDFS home directory required to execute MapReduce jobs exists:

```
hdfs dfs -ls /user/${USER}
```

- Type the following commands:

```
hdfs dfs -ls
hdfs dfs -ls /
hdfs dfs -mkdir test
```

- Create a local file user.txt containing your name and the date:

```
In [1]: # %%bash
        # echo "FirstName LastName" > user.txt
        # echo `date` >> user.txt
        # cat user.txt
```

Copy it on HDFS :

```
hdfs dfs -put user.txt
```

Check with:

```
hdfs dfs -ls -R
hdfs dfs -cat user.txt
hdfs dfs -tail user.txt
```

```
In [2]: # %%bash
        # hdfs dfs -put user.txt
        # hdfs dfs -ls -R /user/navaro_p/
```

```
In [3]: # %%bash
        # hdfs dfs -cat user.txt
```

Remove the file:

```
hdfs dfs -rm user.txt
```

Put it again on HDFS and move to books directory:

```
hdfs dfs -copyFromLocal user.txt
hdfs dfs -mv user.txt books/user.txt
hdfs dfs -ls -R -h
```

Copy user.txt to hello.txt and remove it.

```
hdfs dfs -cp books/user.txt books/hello.txt
hdfs dfs -count -h /user/$USER
hdfs dfs -rm books/user.txt
```

14.6 Hands-on practice:

1. Create a directory `files` in HDFS.
2. List the contents of a directory /.
3. Upload the file today.txt in HDFS.

   ```
   date > today.txt
   whoami >> today.txt
   ```

4. Display contents of file `today.txt`
5. Copy `today.txt` file from source to `files` directory.
6. Copy file `jps.txt` from/To Local file system to HDFS

   ```
   jps > jps.txt
   ```

7. Move file `jps.txt` from source to `files`.
8. Remove file `today.txt` from home directory in HDFS.
9. Display last few lines of `jps.txt`.
10. Display the help of `du` command and show the total amount of space in a human-readable fashion used by your home hdfs directory.
11. Display the help of `df` command and show the total amount of space available in the filesystem in a human-readable fashion.
12. With `chmod` change the rights of `today.txt` file. I has to be readable and writeable only by you.

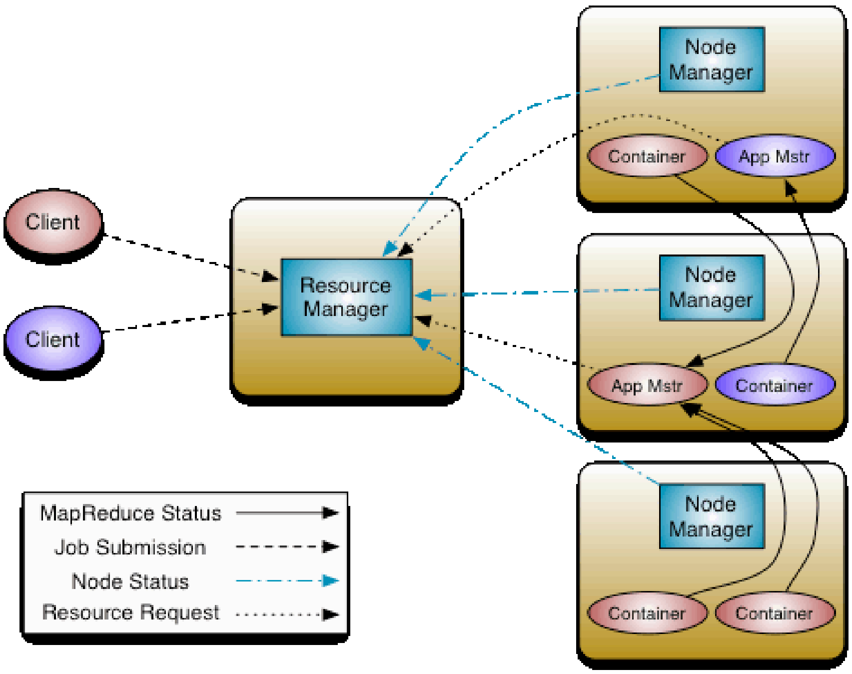
14.7 YARN

YARN takes care of resource management and job scheduling/monitoring.

- The **ResourceManager** is the ultimate authority that arbitrates resources among all the applications in the system. It has two components: **Scheduler** and **ApplicationsManager**.
- The **NodeManager** is the per-machine framework agent who is responsible for **Containers**, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the **ResourceManager/Scheduler**.

The per-application **ApplicationMaster** negotiates resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.

- The **Scheduler** is responsible for allocating resources to the applications.
- The **ApplicationsManager** is responsible for accepting job-submissions, tracking their status and monitoring for progress.



Source: http://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/yarn_architecture.gif

14.7.1 Yarn Web Interface

The JobTracker web UI provides information about general job statistics of the Hadoop cluster, running/completed/failed jobs and a job history log file. It also gives access to the "local machine's" Hadoop log files (the machine on which the web UI is running on).

- All Applications http://svmass2.mass.uhb.fr:8088

14.8 WordCount Example

The Worcount example is implemented in Java and it is the example of Hadoop MapReduce Tutorial

Let's create some files with lorem python package

In [4]:
```
from lorem import text

for i in range(1,10):
    with open('sample{0:02d}.txt'.format(i), 'w') as f:
        f.write(text())
```

- Make input directory in your HDFS home directory required to execute MapReduce jobs:

```
hdfs dfs -mkdir -p /user/${USER}/input
```

`-p` flag force the directory creation even if it already exists.

14.8.1 Exercise

- Copy all necessary files in HDFS system.
- Run the Java example using the command

```
hadoop jar /export/hadoop-2.7.6/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.6.jar wordcount /us
```

- Remove the output directory and try to use yarn

```
yarn jar /export/hadoop-2.7.6/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.6.jar wordcount /user
```

- Connect to the Yarn web user interface and read the logs carefully.

14.9 Deploying the MapReduce Python code on Hadoop

This Python must use the Hadoop Streaming API to pass data between our Map and Reduce code via Python's sys.stdin (standard input) and sys.stdout (standard output).

14.10 Map

The following Python code read data from sys.stdin, split it into words and output a list of lines mapping words to their (intermediate) counts to sys.stdout. For every word it outputs 1 tuples immediately.

```
In [5]: %%file mapper.py

        from __future__ import print_function # for python2 compatibility
        import sys, string
        translator = str.maketrans('', '', string.punctuation)
        # input comes from standard input
        for line in sys.stdin:
            line = line.strip().lower() # remove leading and trailing whitespace
            line = line.translate(translator)   # strip punctuation
            for word in line.split(): # split the line into words
                # write the results to standard output;
                # what we output here will be the input for the
                # Reduce step, i.e. the input for reducer.py
                # tab-delimited; the trivial word count is 1
                print (f'{word}\t 1')
```

```
Overwriting mapper.py
```

```
In [6]: import sys
        with open("mapper.py", "r+") as f:
            s = f.read()
            f.seek(0)
            f.write("#!"+sys.executable+"\n" + s)
```

The python script must be executable:

```
chmod +x mapper.py
```

Try to run in a terminal with:

```
cat sample01.txt | ./mapper.py | sort
```

or

```
./mapper.py < sample01.txt | sort
```

```
In [7]: # %%bash
        # chmod +x mapper.py
        # cat sample01.txt | ./mapper.py | sort
```

14.11 Reduce

The following code reads the results of mapper.py and sum the occurrences of each word to a final count, and then output its results to sys.stdout. Remember that Hadoop sorts map output so it is easier to count words.

```
In [8]: %%file reducer.py
        from __future__ import print_function
        from operator import itemgetter
        import sys

        current_word = None
        current_count = 0
        word = None

        for line in sys.stdin:

            # parse the input we got from mapper.py
            word, count = line.split('\t', 1)

            # convert count (currently a string) to int
            try:
                count = int(count)
            except ValueError:
                # count was not a number, so silently
                # ignore/discard this line
                continue

            # this IF-switch only works because Hadoop sorts map output
            # by key (here: word) before it is passed to the reducer
            if current_word == word:
                current_count += count
            else:
                if current_word:
                    # write result to sys.stdout
                    print (f'{current_count}\t{current_word}')
                current_count = count
                current_word = word

        # do not forget to output the last word if needed!
        if current_word == word:
            print (f'{current_count}\t{current_word}')
```

```
Overwriting reducer.py
```

```
In [9]: import sys
        with open("reducer.py", "r+") as f:
            s = f.read()
            f.seek(0)
            f.write("#!"+sys.executable+"\n" + s)
```

As mapper the python script must be executable:

```
chmod +x reducer.py
```

Try to run in a terminal with:

```
cat sample.txt | ./mapper.py | sort | ./reducer.py | sort
```

or

```
./mapper.py < sample01.txt | sort | ./reducer.py | sort
```

```
In [10]: # %%bash
         # chmod +x reducer.py
         # ./mapper.py < sample01.txt | sort | ./reducer.py | sort
```

14.12 Execution on Hadoop cluster

- Copy all files to HDFS cluster
- Run the WordCount MapReduce

```
In [11]: %%file Makefile

         HADOOP_VERSION=2.7.6
         HADOOP_HOME=/export/hadoop-${HADOOP_VERSION}
         HADOOP_TOOLS=${HADOOP_HOME}/share/hadoop/tools/lib
         HDFS_DIR=/user/${USER}

         SAMPLES = sample01.txt sample02.txt sample03.txt sample04.txt

         copy_to_hdfs: ${SAMPLES}
                 hdfs dfs -mkdir -p ${HDFS_DIR}/input
                 hdfs dfs -put $^ ${HDFS_DIR}/input

         run_with_hadoop:
                 hadoop jar ${HADOOP_TOOLS}/hadoop-streaming-${HADOOP_VERSION}.jar \
             -file  ${PWD}/mapper.py  -mapper  ${PWD}/mapper.py \
             -file  ${PWD}/reducer.py -reducer ${PWD}/reducer.py \
             -input ${HDFS_DIR}/input/*.txt -output ${HDFS_DIR}/output-hadoop

         run_with_yarn:
                 yarn jar ${HADOOP_TOOLS}/hadoop-streaming-${HADOOP_VERSION}.jar \
                 -file  ${PWD}/mapper.py  -mapper  ${PWD}/mapper.py \
                 -file  ${PWD}/reducer.py -reducer ${PWD}/reducer.py \
                 -input ${HDFS_DIR}/input/*.txt -output ${HDFS_DIR}/output-yarn
```

```
Overwriting Makefile
```

```
In [12]: # %%bash
         # hdfs dfs -rm -r input
         # make copy_to_hdfs
         # hdfs dfs -ls input
```

```
In [13]: # %%bash
         # hdfs dfs -rm -r -f output-hadoop
         # make run_with_hadoop
         # hdfs dfs -cat output-hadoop/*
```

Chapter 15

Hadoop File Formats

Format Wars: From VHS and Beta to Avro and Parquet

15.1 Feather

For light data, it is recommanded to use Feather. It is a fast, interoperable data frame storage that comes with bindings for python and R.

Feather uses also the Apache Arrow columnar memory specification to represent binary data on disk. This makes read and write operations very fast.

```
In [1]: import feather
        import pandas as pd
        import numpy as np
        arr = np.random.randn(10000) # 10% nulls
        arr[::10] = np.nan
        df = pd.DataFrame({'column_{0}'.format(i): arr for i in range(10)})
        feather.write_dataframe(df, 'test.feather')
```

```
In [2]: df = pd.read_feather("test.feather")

        df.head()
```

```
Out[2]:    column_0  column_1  column_2  column_3  column_4  column_5  column_6  \
        0       NaN       NaN       NaN       NaN       NaN       NaN       NaN
        1  0.913605  0.913605  0.913605  0.913605  0.913605  0.913605  0.913605
        2  0.124875  0.124875  0.124875  0.124875  0.124875  0.124875  0.124875
        3  0.483494  0.483494  0.483494  0.483494  0.483494  0.483494  0.483494
        4  0.455127  0.455127  0.455127  0.455127  0.455127  0.455127  0.455127

           column_7  column_8  column_9
        0       NaN       NaN       NaN
        1  0.913605  0.913605  0.913605
        2  0.124875  0.124875  0.124875
        3  0.483494  0.483494  0.483494
        4  0.455127  0.455127  0.455127
```

15.2 Parquet file format

Parquet format is a common binary data store, used particularly in the Hadoop/big-data sphere. It provides several advantages relevant to big-data processing:

- columnar storage, only read the data of interest
- efficient binary packing
- choice of compression algorithms and encoding
- split data into files, allowing for parallel processing
- range of logical types
- statistics stored in metadata allow for skipping unneeded chunks
- data partitioning using the directory structure

15.3 Apache Arrow

Arrow is a columnar in-memory analytics layer designed to accelerate big data. It houses a set of canonical in-memory representations of flat and hierarchical data along with multiple language-bindings for structure manipulation.

https://arrow.apache.org/docs/python/parquet.html

The Apache Parquet project provides a standardized open-source columnar storage format for use in data analysis systems. It was created originally for use in Apache Hadoop with systems like Apache Drill, Apache Hive, Apache Impala, and Apache Spark adopting it as a shared standard for high performance data IO.

Apache Arrow is an ideal in-memory transport layer for data that is being read or written with Parquet files. PyArrow includes Python bindings to read and write Parquet files with pandas.

Example:

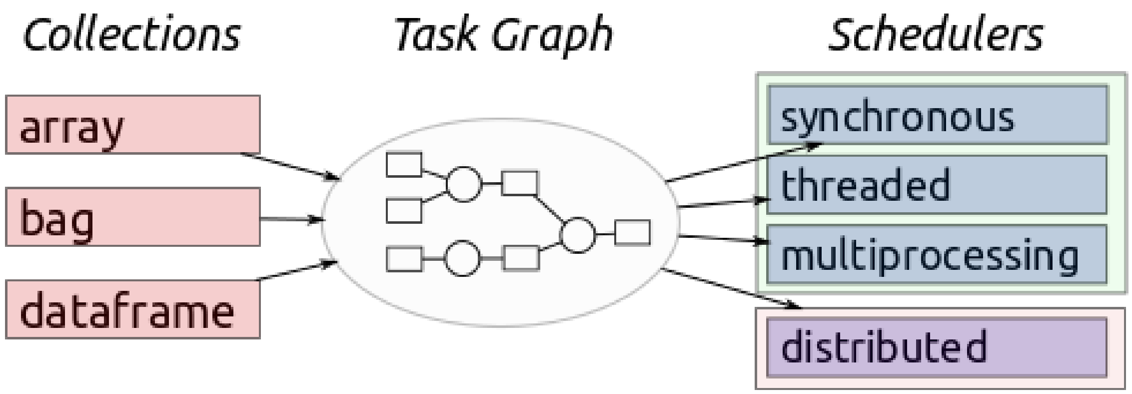
```
import pyarrow as pa

hdfs = pa.hdfs.connect('svmass2.mass.uhb.fr', 54311, 'navaro_p')
```

```
In [3]: import pyarrow.parquet as pq
        import numpy as np
        import pandas as pd
        import pyarrow as pa

In [4]: df = pd.DataFrame({'one': [-1, np.nan, 2.5],
                           'two': ['foo', 'bar', 'baz'],
                           'three': [True, False, True]},
                           index=list('abc'))
        table = pa.Table.from_pandas(df)

In [5]: pq.write_table(table, 'example.parquet')

In [6]: table2 = pq.read_table('example.parquet')

In [7]: table2.to_pandas()

Out[7]:     one  two  three
         a -1.0  foo   True
         b  NaN  bar  False
         c  2.5  baz   True

In [8]: pq.read_table('example.parquet', columns=['one', 'three'])

Out[8]: pyarrow.Table
        one: double
        three: bool

In [9]: pq.read_pandas('example.parquet', columns=['two']).to_pandas()

Out[9]:     two
         a  foo
         b  bar
         c  baz
```

Chapter 16

Dask Dataframes

Dask is a flexible parallel computing library for analytic computing written in Python. Dask is similar to Spark, by lazily constructing directed acyclic graph (DAG) of tasks and splitting large datasets into small portions called partitions. See the below image from Dask's web page for illustration.



It has three main interfaces:

- Array, which works like NumPy arrays;
- Bag, which is similar to RDD interface in Spark;
- DataFrame, which works like Pandas DataFrame.

While it can work on a distributed cluster, Dask works also very well on a single cpu machine.

16.1 DataFrames

Dask dataframes look and feel (mostly) like Pandas dataframes but they run on the same infrastructure that powers dask.delayed.

The `dask.dataframe` module implements a blocked parallel `DataFrame` object that mimics a large subset of the Pandas `DataFrame`. One dask `DataFrame` is comprised of many in-memory pandas `DataFrames` separated along the index. One operation on a dask `DataFrame` triggers many pandas operations on the constituent pandas `DataFrames` in a way that is mindful of potential parallelism and memory constraints.

Related Documentation

- Dask DataFrame documentation
- Pandas documentation

In this notebook, we will extracts some historical flight data for flights out of NYC between 1990 and 2000. The data is taken from here. This should only take a few seconds to run.

We will use `dask.dataframe` construct our computations for us. The `dask.dataframe.read_csv` function can take a globstring like `"data/nycflights/*.csv"` and build parallel computations on all of our data at once.

16.1.1 Prep the Data

```
In [1]: import os
        import pandas as pd
        pd.set_option("max.rows", 10)
        os.getcwd()

Out[1]: '/home/runner/work/big-data/big-data/notebooks'
```

```
In [2]: import os  # library to get directory and file paths
        import tarfile # this module makes possible to read and write tar archives

        def extract_flight():
            here = os.getcwd()
            flightdir = os.path.join(here,'data', 'nycflights')
            if not os.path.exists(flightdir):
               print("Extracting flight data")
               tar_path = os.path.join('data', 'nycflights.tar.gz')
               with tarfile.open(tar_path, mode='r:gz') as flights:
                  flights.extractall('data/')

        extract_flight() # this function call will extract 10 csv files in data/nycflights

Extracting flight data
```

16.1.2 Load Data from CSVs in Dask Dataframes

```
In [3]: import os
        here = os.getcwd()
        filename = os.path.join(here, 'data', 'nycflights', '*.csv')
        filename

Out[3]: '/home/runner/work/big-data/big-data/notebooks/data/nycflights/*.csv'
```

```
In [4]: import dask
        import dask.dataframe as dd

        df = dd.read_csv(filename,
                         parse_dates={'Date': [0, 1, 2]})
```

Let’s take a look to the dataframe

```
In [5]: df

Out[5]: Dask DataFrame Structure:
                                   Date DayOfWeek  DepTime CRSDepTime  ArrTime CRSArrTime UniqueCarrier
        npartitions=10
                         datetime64[ns]     int64  float64      int64  float64      int64        object
                                    ...       ...      ...        ...      ...        ...           ...    ...    ...
        ...                         ...       ...      ...        ...      ...        ...           ...    ...    ...
                                    ...       ...      ...        ...      ...        ...           ...    ...    ...
                                    ...       ...      ...        ...      ...        ...           ...    ...    ...
        Dask Name: read-csv, 10 tasks
```

```
In [6]: ### Get the first 5 rows
        df.head()

Out[6]:          Date  DayOfWeek  DepTime  CRSDepTime  ArrTime  CRSArrTime  \
        0 1990-01-01          1   1621.0        1540   1747.0        1701
        1 1990-01-02          2   1547.0        1540   1700.0        1701
        2 1990-01-03          3   1546.0        1540   1710.0        1701
        3 1990-01-04          4   1542.0        1540   1710.0        1701
        4 1990-01-05          5   1549.0        1540   1706.0        1701

          UniqueCarrier  FlightNum  TailNum  ActualElapsedTime  …  AirTime  \
        0            US         33      NaN               86.0  …      NaN
        1            US         33      NaN               73.0  …      NaN
        2            US         33      NaN               84.0  …      NaN
        3            US         33      NaN               88.0  …      NaN
        4            US         33      NaN               77.0  …      NaN

           ArrDelay  DepDelay  Origin Dest Distance  TaxiIn  TaxiOut  Cancelled  \
        0      46.0      41.0     EWR  PIT    319.0     NaN      NaN          0
        1      -1.0       7.0     EWR  PIT    319.0     NaN      NaN          0
        2       9.0       6.0     EWR  PIT    319.0     NaN      NaN          0
        3       9.0       2.0     EWR  PIT    319.0     NaN      NaN          0
        4       5.0       9.0     EWR  PIT    319.0     NaN      NaN          0

           Diverted
        0         0
        1         0
        2         0
        3         0
        4         0

        [5 rows x 21 columns]

In [7]: import traceback # we use traceback because we except an error.

        try:
            df.tail() # Get the last 5 rows
        except Exception:
            traceback.print_exc()

Traceback (most recent call last):
  File "<ipython-input-7-7cb27b738c02>", line 4, in <module>
    df.tail() # Get the last 5 rows
  File "/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/dataframe/core.py", line 1
    result = result.compute()
  File "/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/base.py", line 167, in com
    (result,) = compute(self, traverse=False, **kwargs)
  File "/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/base.py", line 447, in com
    results = schedule(dsk, keys, **kwargs)
  File "/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/threaded.py", line 76, in
    results = get_async(
  File "/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/local.py", line 486, in ge
    raise_exception(exc, tb)
  File "/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/local.py", line 316, in re
    raise exc
  File "/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/local.py", line 222, in ex
```

```
    result = _execute_task(task, data)
  File "/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/core.py", line 121, in _exe
    return func(*(_execute_task(a, cache) for a in args))
  File "/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/core.py", line 121, in <ge
    return func(*(_execute_task(a, cache) for a in args))
  File "/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/core.py", line 121, in _exe
    return func(*(_execute_task(a, cache) for a in args))
  File "/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/dataframe/io/csv.py", line
    coerce_dtypes(df, dtypes)
  File "/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/dataframe/io/csv.py", line
    raise ValueError(msg)
ValueError: Mismatched dtypes found in `pd.read_csv`/`pd.read_table`.

+----------------+---------+----------+
| Column         | Found   | Expected |
+----------------+---------+----------+
| CRSElapsedTime | float64 | int64    |
| TailNum        | object  | float64  |
+----------------+---------+----------+

The following columns also raised exceptions on conversion:

- TailNum
  ValueError("could not convert string to float: 'N54711'")

Usually this is due to dask's dtype inference failing, and
*may* be fixed by specifying dtypes manually by adding:

dtype={'CRSElapsedTime': 'float64',
       'TailNum': 'object'}

to the call to `read_csv`/`read_table`.
```

16.1.3 What just happened?

Unlike `pandas.read_csv` which reads in the entire file before inferring datatypes, `dask.dataframe.read_csv` only reads in a sample from the beginning of the file (or first file if using a glob). These inferred datatypes are then enforced when reading all partitions.

In this case, the datatypes inferred in the sample are incorrect. The first `n` rows have no value for `CRSElapsedTime` (which pandas infers as a `float`), and later on turn out to be strings (`object` dtype). When this happens you have a few options:

- Specify dtypes directly using the `dtype` keyword. This is the recommended solution, as it's the least error prone (better to be explicit than implicit) and also the most performant.
- Increase the size of the `sample` keyword (in bytes)
- Use `assume_missing` to make `dask` assume that columns inferred to be `int` (which don't allow missing values) are actually floats (which do allow missing values). In our particular case this doesn't apply.

In our case we'll use the first option and directly specify the `dtypes` of the offending columns.

In [8]:
```
df.dtypes
```

Out[8]:
```
Date          datetime64[ns]
DayOfWeek              int64
```

```
DepTime              float64
CRSDepTime             int64
ArrTime              float64
                      …
Distance             float64
TaxiIn               float64
TaxiOut              float64
Cancelled              int64
Diverted               int64
Length: 21, dtype: object
```

```
In [9]: df = dd.read_csv(filename,
                        parse_dates={'Date': [0, 1, 2]},
                        dtype={'TailNum': object,
                               'CRSElapsedTime': float,
                               'Cancelled': bool})
```

```
In [10]: df.tail()
```

```
Out[10]:             Date  DayOfWeek  DepTime  CRSDepTime  ArrTime  CRSArrTime  \
        269176 1999-12-27          1   1645.0        1645   1830.0        1901
        269177 1999-12-28          2   1726.0        1645   1928.0        1901
        269178 1999-12-29          3   1646.0        1645   1846.0        1901
        269179 1999-12-30          4   1651.0        1645   1908.0        1901
        269180 1999-12-31          5   1642.0        1645   1851.0        1901

               UniqueCarrier  FlightNum TailNum  ActualElapsedTime  …  AirTime  \
        269176            UA       1753  N516UA              225.0  …    205.0
        269177            UA       1753  N504UA              242.0  …    214.0
        269178            UA       1753  N592UA              240.0  …    220.0
        269179            UA       1753  N575UA              257.0  …    233.0
        269180            UA       1753  N539UA              249.0  …    232.0

                ArrDelay  DepDelay  Origin Dest Distance  TaxiIn  TaxiOut  Cancelled  \
        269176     -31.0       0.0     LGA  DEN   1619.0     7.0     13.0      False
        269177      27.0      41.0     LGA  DEN   1619.0     5.0     23.0      False
        269178     -15.0       1.0     LGA  DEN   1619.0     5.0     15.0      False
        269179       7.0       6.0     LGA  DEN   1619.0     5.0     19.0      False
        269180     -10.0      -3.0     LGA  DEN   1619.0     6.0     11.0      False

                Diverted
        269176         0
        269177         0
        269178         0
        269179         0
        269180         0

        [5 rows x 21 columns]
```

Let's take a look at one more example to fix ideas.

```
In [11]: len(df)
```

```
Out[11]: 2611892
```

16.1.4 Why df is ten times longer ?

- Dask investigated the input path and found that there are ten matching files.
- A set of jobs was intelligently created for each chunk - one per original CSV file in this case.
- Each file was loaded into a pandas dataframe, had `len()` applied to it.
- The subtotals were combined to give you the final grant total.

16.2 Computations with `dask.dataframe`

We compute the maximum of the `DepDelay` column. With `dask.delayed` we could create this computation as follows:

```
maxes = []
for fn in filenames:
    df = dask.delayed(pd.read_csv)(fn)
    maxes.append(df.DepDelay.max())

final_max = dask.delayed(max)(maxes)
final_max.compute()
```

Now we just use the normal Pandas syntax as follows:

```
In [12]: %time df.DepDelay.max().compute()
```

```
CPU times: user 3.99 s, sys: 450 ms, total: 4.44 s
Wall time: 3 s
```

```
Out[12]: 1435.0
```

This writes the delayed computation for us and then runs it. Recall that the delayed computation is a dask graph made of up of key-value pairs.

Some things to note:

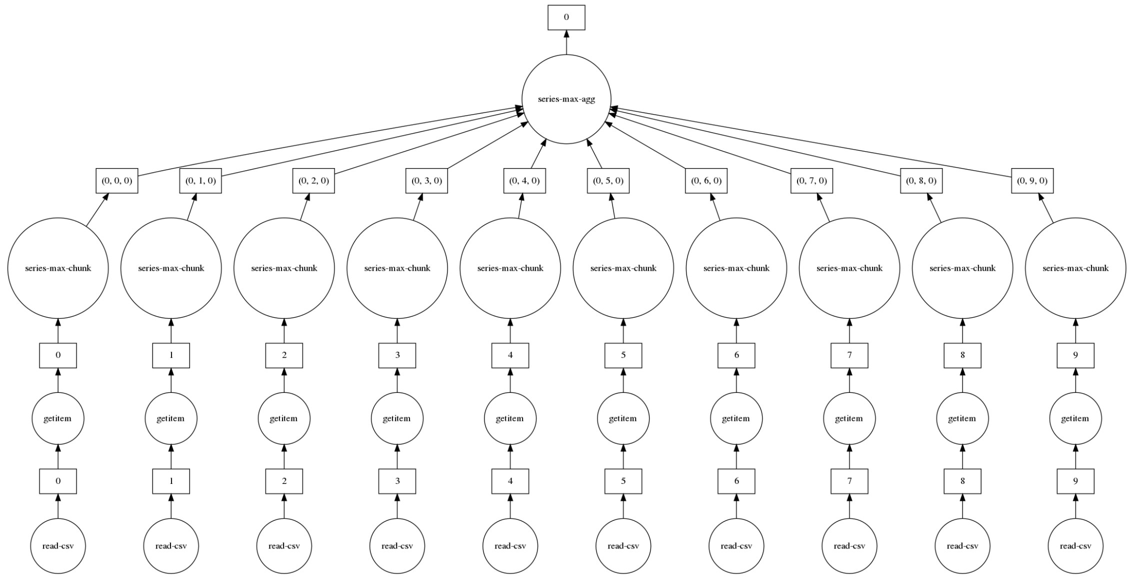
1. As with `dask.delayed`, we need to call `.compute()` when we're done. Up until this point everything is lazy.
2. Dask will delete intermediate results (like the full pandas dataframe for each file) as soon as possible.

   - This lets us handle datasets that are larger than memory
   - This means that repeated computations will have to load all of the data in each time (run the code above again, is it faster or slower than you would expect?)

As with `Delayed` objects, you can view the underlying task graph using the `.visualize` method:

```
In [13]: df.DepDelay.max().visualize()
```

```
Out[13]:
```



If you are already familiar with the Pandas API then know how to use `dask.dataframe`. There are a couple of small changes.

As noted above, computations on dask `DataFrame` objects don't perform work, instead they build up a dask graph. We can evaluate this dask graph at any time using the `.compute()` method.

```
In [14]: result = df.DepDelay.mean()  # create the tasks graph
```

```
In [15]: %time result.compute()          # perform actual computation
```

```
CPU times: user 4.21 s, sys: 543 ms, total: 4.75 s
Wall time: 3.21 s
```

```
Out[15]: 9.206602541321965
```

16.3 Store Data in Apache Parquet Format

Dask encourage dataframe users to store and load data using Parquet instead. Apache Parquet is a columnar binary format that is easy to split into multiple files (easier for parallel loading) and is generally much simpler to deal with than HDF5 (from the Dask library's perspective). It is also a common format used by other big data systems like Apache Spark and Apache Impala and so is useful to interchange with other systems.

```
In [16]: df.drop("TailNum", axis=1).to_parquet("nycflights/")  # save csv files using parquet format
```

It is possible to specify dtypes and compression when converting. This can definitely help give you significantly greater speedups, but just using the default settings will still be a large improvement.

```
In [17]: df.size.compute()
```

```
Out[17]: 54849732
```

```
In [18]: import dask.dataframe as dd
         df = dd.read_parquet("nycflights/")
         df.head()
```

```
Out[18]:          Date  DayOfWeek  DepTime  CRSDepTime  ArrTime  CRSArrTime  \
         0 1990-01-01          1   1621.0        1540   1747.0        1701
         1 1990-01-02          2   1547.0        1540   1700.0        1701
         2 1990-01-03          3   1546.0        1540   1710.0        1701
         3 1990-01-04          4   1542.0        1540   1710.0        1701
         4 1990-01-05          5   1549.0        1540   1706.0        1701

           UniqueCarrier  FlightNum  ActualElapsedTime  CRSElapsedTime  AirTime  \
         0            US         33               86.0            81.0      NaN
         1            US         33               73.0            81.0      NaN
         2            US         33               84.0            81.0      NaN
         3            US         33               88.0            81.0      NaN
         4            US         33               77.0            81.0      NaN

            ArrDelay  DepDelay Origin Dest  Distance  TaxiIn  TaxiOut  Cancelled  \
         0      46.0      41.0    EWR  PIT     319.0     NaN      NaN      False
         1      -1.0       7.0    EWR  PIT     319.0     NaN      NaN      False
         2       9.0       6.0    EWR  PIT     319.0     NaN      NaN      False
         3       9.0       2.0    EWR  PIT     319.0     NaN      NaN      False
         4       5.0       9.0    EWR  PIT     319.0     NaN      NaN      False

            Diverted
         0         0
         1         0
         2         0
         3         0
         4         0
```

```
In [19]: result = df.DepDelay.mean()
```

```
In [20]: %time result.compute()
```

```
CPU times: user 133 ms, sys: 17 ms, total: 150 ms
Wall time: 106 ms
```

```
Out[20]: 9.206602541321965
```

The computation is much faster because pulling out the DepDelay column is easy for Parquet.

16.3.1 Parquet advantages:

- Binary representation of data, allowing for speedy conversion of bytes-on-disk to bytes-in-memory
- Columnar storage, meaning that you can load in as few columns as you need without loading the entire dataset
- Row-chunked storage so that you can pull out data from a particular range without touching the others
- Per-chunk statistics so that you can find subsets quickly
- Compression

16.3.2 Exercise 15.1

If you don't remember how to use pandas. Please read pandas documentation.

- Use the `head()` method to get the first ten rows
- How many rows are in our dataset?

- Use selections `df[...]` to find how many positive (late) and negative (early) departure times there are
- In total, how many non-cancelled flights were taken? (To invert a boolean pandas Series s, use ~s).

```
In [21]: df.head(10)
```

```
Out[21]:         Date  DayOfWeek  DepTime  CRSDepTime  ArrTime  CRSArrTime  \
         0 1990-01-01          1   1621.0        1540   1747.0        1701
         1 1990-01-02          2   1547.0        1540   1700.0        1701
         2 1990-01-03          3   1546.0        1540   1710.0        1701
         3 1990-01-04          4   1542.0        1540   1710.0        1701
         4 1990-01-05          5   1549.0        1540   1706.0        1701
         5 1990-01-06          6   1539.0        1540   1653.0        1701
         6 1990-01-07          7   1553.0        1540   1713.0        1701
         7 1990-01-08          1   1543.0        1540   1656.0        1701
         8 1990-01-09          2   1540.0        1540   1704.0        1701
         9 1990-01-10          3   1608.0        1540   1740.0        1701

           UniqueCarrier  FlightNum  ActualElapsedTime  CRSElapsedTime  AirTime  \
         0            US         33               86.0            81.0      NaN
         1            US         33               73.0            81.0      NaN
         2            US         33               84.0            81.0      NaN
         3            US         33               88.0            81.0      NaN
         4            US         33               77.0            81.0      NaN
         5            US         33               74.0            81.0      NaN
         6            US         33               80.0            81.0      NaN
         7            US         33               73.0            81.0      NaN
         8            US         33               84.0            81.0      NaN
         9            US         33               92.0            81.0      NaN

            ArrDelay  DepDelay Origin Dest  Distance  TaxiIn  TaxiOut  Cancelled  \
         0      46.0      41.0    EWR  PIT     319.0     NaN      NaN      False
         1      -1.0       7.0    EWR  PIT     319.0     NaN      NaN      False
         2       9.0       6.0    EWR  PIT     319.0     NaN      NaN      False
         3       9.0       2.0    EWR  PIT     319.0     NaN      NaN      False
         4       5.0       9.0    EWR  PIT     319.0     NaN      NaN      False
         5      -8.0      -1.0    EWR  PIT     319.0     NaN      NaN      False
         6      12.0      13.0    EWR  PIT     319.0     NaN      NaN      False
         7      -5.0       3.0    EWR  PIT     319.0     NaN      NaN      False
         8       3.0       0.0    EWR  PIT     319.0     NaN      NaN      False
         9      39.0      28.0    EWR  PIT     319.0     NaN      NaN      False

            Diverted
         0         0
         1         0
         2         0
         3         0
         4         0
         5         0
         6         0
         7         0
         8         0
         9         0
```

```
In [22]: len(df)
```

```
Out[22]: 2611892

In [23]: len(df[df.DepDelay > 0])

Out[23]: 1187146

In [24]: len(df[df.DepDelay < 0])

Out[24]: 840942

In [25]: len(df[~df.Cancelled])

Out[25]: 2540961
```

16.4 Divisions and the Index

The Pandas index associates a value to each record/row of your data. Operations that align with the index, like `loc` can be a bit faster as a result.

In `dask.dataframe` this index becomes even more important. Recall that one dask `DataFrame` consists of several Pandas `DataFrames`. These dataframes are separated along the index by value. For example, when working with time series we may partition our large dataset by month.

Recall that these many partitions of our data may not all live in memory at the same time, instead they might live on disk; we simply have tasks that can materialize these pandas `DataFrames` on demand.

Partitioning your data can greatly improve efficiency. Operations like `loc`, `groupby`, and `merge/join` along the index are *much more efficient* than operations along other columns. You can see how your dataset is partitioned with the `.divisions` attribute. Note that data that comes out of simple data sources like CSV files aren't intelligently indexed by default. In these cases the values for `.divisions` will be `None`.

```
In [26]: df = dd.read_csv(filename,
                          dtype={'TailNum': str,
                                 'CRSElapsedTime': float,
                                 'Cancelled': bool})
         df.divisions

Out[26]: (None, None, None, None, None, None, None, None, None, None, None)
```

However if we set the index to some new column then dask will divide our data roughly evenly along that column and create new divisions for us. Warning, `set_index` triggers immediate computation.

```
In [27]: df2 = df.set_index('Year')
         df2.divisions

Out[27]: (1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 1999)
```

We see here the minimum and maximum values (1990 and 1999) as well as the intermediate values that separate our data well. This dataset has ten partitions, as the final value is assumed to be the inclusive right-side for the last bin.

```
In [28]: df2.npartitions

Out[28]: 10

In [29]: df2.head()

Out[29]:       Month  DayofMonth  DayOfWeek  DepTime  CRSDepTime  ArrTime  CRSArrTime  \
         Year
         1990      1           1          1   1621.0        1540   1747.0        1701
         1990      1           2          2   1547.0        1540   1700.0        1701
```

```
1990      1           3          3   1546.0        1540   1710.0         1701
1990      1           4          4   1542.0        1540   1710.0         1701
1990      1           5          5   1549.0        1540   1706.0         1701

     UniqueCarrier  FlightNum TailNum  …  AirTime  ArrDelay  DepDelay  \
Year                                   …
1990            US         33     NaN  …      NaN      46.0      41.0
1990            US         33     NaN  …      NaN      -1.0       7.0
1990            US         33     NaN  …      NaN       9.0       6.0
1990            US         33     NaN  …      NaN       9.0       2.0
1990            US         33     NaN  …      NaN       5.0       9.0

      Origin  Dest Distance TaxiIn  TaxiOut  Cancelled  Diverted
Year
1990     EWR   PIT    319.0    NaN      NaN      False         0
1990     EWR   PIT    319.0    NaN      NaN      False         0
1990     EWR   PIT    319.0    NaN      NaN      False         0
1990     EWR   PIT    319.0    NaN      NaN      False         0
1990     EWR   PIT    319.0    NaN      NaN      False         0

[5 rows x 22 columns]
```

One of the benefits of this is that operations like `loc` only need to load the relevant partitions

```
In [30]: df2.loc[1991]

Out[30]: Dask DataFrame Structure:
                        Month DayofMonth DayOfWeek  DepTime CRSDepTime  ArrTime CRSArrTime UniqueCarrie
         npartitions=1
         1991           int64      int64     int64  float64      int64  float64      int64         objec
         1991             …          …         …        …          …        …          …             …         …     …
         Dask Name: loc, 31 tasks

In [31]: df2.loc[1991].compute()

Out[31]:       Month  DayofMonth  DayOfWeek  DepTime  CRSDepTime  ArrTime  CRSArrTime  \
         Year
         1991      1           8          2   1215.0        1215   1340.0         1336
         1991      1           9          3   1215.0        1215   1353.0         1336
         1991      1          10          4   1216.0        1215   1332.0         1336
         1991      1          11          5   1303.0        1215   1439.0         1336
         1991      1          12          6   1215.0        1215   1352.0         1336
         …        …          …          …        …          …        …           …
         1991     12          26          4   1600.0        1600   1857.0         1906
         1991     12          27          5   1600.0        1600   1853.0         1906
         1991     12          28          6   1600.0        1600   1856.0         1906
         1991     12          29          7   1601.0        1600   1851.0         1906
         1991     12          31          2   1558.0        1600   1851.0         1906

              UniqueCarrier  FlightNum TailNum  …  AirTime  ArrDelay  DepDelay  \
         Year                                   …
         1991            US        121     NaN  …      NaN       4.0       0.0
         1991            US        121     NaN  …      NaN      17.0       0.0
         1991            US        121     NaN  …      NaN      -4.0       1.0
         1991            US        121     NaN  …      NaN      63.0      48.0
```

```
1991            US        121     NaN  …       NaN      16.0        0.0
…            …       …      …   …       …        …       …
1991            CO       1539     NaN  …       NaN      -9.0        0.0
1991            CO       1539     NaN  …       NaN     -13.0        0.0
1991            CO       1539     NaN  …       NaN     -10.0        0.0
1991            CO       1539     NaN  …       NaN     -15.0        1.0
1991            CO       1539     NaN  …       NaN     -15.0       -2.0

      Origin  Dest Distance TaxiIn  TaxiOut  Cancelled  Diverted
Year
1991     EWR   PIT    319.0    NaN      NaN      False         0
1991     EWR   PIT    319.0    NaN      NaN      False         0
1991     EWR   PIT    319.0    NaN      NaN      False         0
1991     EWR   PIT    319.0    NaN      NaN      False         0
1991     EWR   PIT    319.0    NaN      NaN      False         0
…       …   …       …     …      …        …       …
1991     LGA   FLL   1076.0    NaN      NaN      False         0
1991     LGA   FLL   1076.0    NaN      NaN      False         0
1991     LGA   FLL   1076.0    NaN      NaN      False         0
1991     LGA   FLL   1076.0    NaN      NaN      False         0
1991     LGA   FLL   1076.0    NaN      NaN      False         0

[258274 rows x 22 columns]
```

16.4.1 Exercises 15.2

In this section we do a few `dask.dataframe` computations. If you are comfortable with Pandas then these should be familiar. You will have to think about when to call `compute`.

- In total, how many non-cancelled flights were taken from each airport?

Hint: use `df.groupby`. `df.groupby(df.A).B.func()`.

- What was the average departure delay from each airport?

Note, this is the same computation you did in the previous notebook (is this approach faster or slower?)

- What day of the week has the worst average departure delay?

In [32]:
```
df = dd.read_parquet("nycflights/")
```

In [33]:
```
df[~df.Cancelled].groupby("Origin").Origin.count().compute()
```

Out[33]:
```
Origin
EWR    1139451
JFK     427243
LGA     974267
Name: Origin, dtype: int64
```

In [34]:
```
df[~df.Cancelled].groupby("Origin").DepDelay.count().compute()
```

Out[34]:
```
Origin
EWR    1139451
JFK     427243
LGA     974267
Name: DepDelay, dtype: int64
```

16.5 Sharing Intermediate Results

When computing all of the above, we sometimes did the same operation more than once. For most operations, `dask.dataframe` hashes the arguments, allowing duplicate computations to be shared, and only computed once.

For example, lets compute the mean and standard deviation for departure delay of all non-cancelled flights:

```
In [35]: non_cancelled = df[~df.Cancelled]
         mean_delay = non_cancelled.DepDelay.mean()
         std_delay = non_cancelled.DepDelay.std()
```

Using two calls to `.compute`:

```
In [36]: %%time
         mean_delay_res = mean_delay.compute()
         std_delay_res = std_delay.compute()
```

```
CPU times: user 2.71 s, sys: 143 ms, total: 2.85 s
Wall time: 1.96 s
```

Using one call to `dask.compute`:

```
In [37]: %%time
         mean_delay_res, std_delay_res = dask.compute(mean_delay, std_delay)
```

```
CPU times: user 1.39 s, sys: 79.3 ms, total: 1.47 s
Wall time: 1.02 s
```

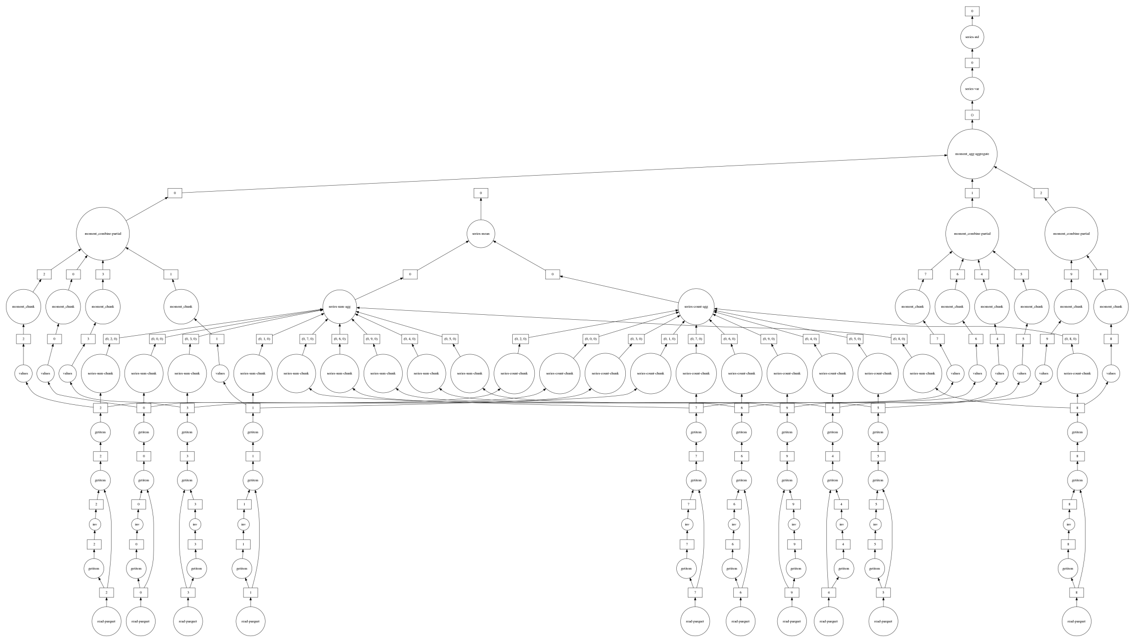
Using `dask.compute` takes roughly 1/2 the time. This is because the task graphs for both results are merged when calling `dask.compute`, allowing shared operations to only be done once instead of twice. In particular, using `dask.compute` only does the following once:

- the calls to `read_csv`
- the filter (`df[~df.Cancelled]`)
- some of the necessary reductions (`sum`, `count`)

To see what the merged task graphs between multiple results look like (and what's shared), you can use the `dask.visualize` function (we might want to use `filename='graph.pdf'` to zoom in on the graph better):

```
In [38]: dask.visualize(mean_delay, std_delay)
```

```
Out[38]:
```



Chapter 17

Spark DataFrames

- Enable wider audiences beyond "Big Data" engineers to leverage the power of distributed processing
- Inspired by data frames in R and Python (Pandas)
- Designed from the ground-up to support modern big data and data science applications
- Extension to the existing RDD API

17.1 References

- Spark SQL, DataFrames and Datasets Guide
- Introduction to DataFrames - Python
- PySpark Cheat Sheet: Spark DataFrames in Python

17.1.1 DataFrames are :

- The preferred abstraction in Spark
- Strongly typed collection of distributed elements
- Built on Resilient Distributed Datasets (RDD)
- Immutable once constructed

17.1.2 With Dataframes you can :

- Track lineage information to efficiently recompute lost data
- Enable operations on collection of elements in parallel

17.1.3 You construct DataFrames

- by parallelizing existing collections (e.g., Pandas DataFrames)
- by transforming an existing DataFrames
- from files in HDFS or any other storage system (e.g., Parquet)

17.1.4 Features

- Ability to scale from kilobytes of data on a single laptop to petabytes on a large cluster
- Support for a wide array of data formats and storage systems
- Seamless integration with all big data tooling and infrastructure via Spark
- APIs for Python, Java, Scala, and R

17.1.5 DataFrames versus RDDs

- Nice API for new users familiar with data frames in other programming languages.

- For existing Spark users, the API will make Spark easier to program than using RDDs
- For both sets of users, DataFrames will improve performance through intelligent optimizations and code-generation

17.2 PySpark Shell

Run the Spark shell:

```
pyspark
```

Output similar to the following will be displayed, followed by a >>> REPL prompt:

```
Python 3.6.5 |Anaconda, Inc.| (default, Apr 29 2018, 16:14:56)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
2018-09-18 17:13:13 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your platform.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.3.1
      /_/

Using Python version 3.6.5 (default, Apr 29 2018 16:14:56)
SparkSession available as 'spark'.
>>>
```

Read data and convert to Dataset

```
df = sqlContext.read.csv("/tmp/irmar.csv", sep=';', header=True)
```

```
>>> df2.show()
+---+--------------------+------------+------+------------+--------+-----+---------+--------+
|_c0|                name|       phone|office|organization|position|  hdr|    team1|   team2|
+---+--------------------+------------+------+------------+--------+-----+---------+--------+
|  0|      Alphonse Paul |+33223235223|   214|          R1|     DOC|False|      EDP|      NA|
|  1|        Ammari Zied |+33223235811|   209|          R1|      MC| True|      EDP|      NA|
.
.
.
| 18|    Bernier Joachim |+33223237558|   214|          R1|     DOC|False|   ANANUM|      NA|
| 19|   Berthelot Pierre |+33223236043|   601|          R1|      PE| True|       GA|      NA|
+---+--------------------+------------+------+------------+--------+-----+---------+--------+
only showing top 20 rows
```

17.3 Transformations, Actions, Laziness

Like RDDs, DataFrames are lazy. Transformations contribute to the query plan, but they don't execute anything. Actions cause the execution of the query.

17.3.1 Transformation examples

- filter
- select
- drop
- intersect
- join ### Action examples
- count
- collect
- show
- head
- take

17.4 Creating a DataFrame in Python

```
In [1]: import sys, subprocess
        import os

        os.environ["PYSPARK_PYTHON"] = sys.executable

In [2]: from pyspark import SparkContext, SparkConf, SQLContext
        # The following three lines are not necessary
        # in the pyspark shell
        conf = SparkConf().setAppName("people").setMaster("local[*]")
        sc = SparkContext(conf=conf)
        sc.setLogLevel("ERROR")
        sqlContext = SQLContext(sc)

In [3]: df = sqlContext.read.json("data/people.json") # get a dataframe from json file

        df.show(24)
```

```
+-----------+---------+-----------+
| firstname| lastname|      login|
+-----------+---------+-----------+
|     Simon|     Uzel|     uzel_s|
|   Perrine|   Moreau|   moreau_p|
|     Elise|    Negri|    negri_e|
|   Camille|   Cochet|   cochet_c|
|   Nolwenn| Giguelay| giguelay_n|
|     Youen|    Meyer|    meyer_y|
|    Emilie|  Lacoste|  lacoste_e|
|       Pia|  LeBihan|  lebihan_p|
|      Yann|    Evain|    evain_y|
|   Camille|    Guyon|    guyon_c|
|  Mathilde|  LeMener|  lemener_m|
|    Gildas| LeGuilly| liguilly_g|
|    Pierre| Gardelle| gardelle_p|
|Christophe|Boulineau|boulineau_c|
|      Omar| Aitichou| aitichou_o|
|     Lijun|      Chi|      chi_l|
|    Jiawei|      Liu|      lin_j|
|     Irvin|Keraudren|keraudren_i|
|     Bryan|    Jacob|    jacob_b|
|   Raphael| Guillerm| guillerm_r|
|     Bruno|Queguiner|queguiner_b|
```

```
|   Yingshi|     Zeng|     zeng_y|
+----------+---------+-----------+
```

17.5 Schema Inference

In this exercise, let's explore schema inference. We're going to be using a file called `irmar.txt`. The data is structured, but it has no self-describing schema. And, it's not JSON, so Spark can't infer the schema automatically. Let's create an RDD and look at the first few rows of the file.

```
In [4]: rdd = sc.textFile("data/irmar.csv")
        for line in rdd.take(10):
          print(line)
```

```
Alphonse Paul;+33223235223;214;R1;DOC;False;EDP;NA
Ammari Zied;+33223235811;209;R1;MC;True;EDP;NA
André Simon;+33223237555;301;R1;DOC;False;THEO-ERG;NA
Angst Jurgen;+33223236519;320;R1;MC;False;PROC-STOC;NA
Bailleul Ismaël;+33223236369;302;R1;MC;True;THEO-ERG;NA
Baker Mark;+33223236028;835;R1;PR;True;GAN;NA
Balac Stephane;+33223236274;110;R1;MC;False;ANANUM;NA
Bauer Max;+33223236675;734;R1;MC;False;GAN;NA
Bavard Juliette;+33223236724;331;CNRS;CR;False;GAN;THEO-ERG
Beauchard Karine;+33223236164;235;R1;PR;True;ANANUM;NA
```

17.6 Hands-on Exercises

You can look at the DataFrames API documentation

Let's take a look to file "/tmp/irmar.csv". Each line consists of the same information about a person:

- name
- phone
- office
- organization
- position
- hdr
- team1
- team2

```
In [5]: from collections import namedtuple

        rdd = sc.textFile("data/irmar.csv")

        Person = namedtuple('Person', ['name', 'phone', 'office', 'organization',
                                       'position', 'hdr', 'team1', 'team2'])
        def str_to_bool(s):
            if s == 'True': return True
            return False

        def map_to_person(line):
            cols = line.split(";")
            return Person(name         = cols[0],
                          phone        = cols[1],
```

```
                          office       = cols[2],
                          organization = cols[3],
                          position     = cols[4],
                          hdr          = str_to_bool(cols[5]),
                          team1        = cols[6],
                          team2        = cols[7])

        people_rdd = rdd.map(map_to_person)
        df = people_rdd.toDF()
```

In [6]: `df.show()`

```
+--------------------+------------+------+------------+--------+-----+---------+--------+
|                name|       phone|office|organization|position|  hdr|    team1|   team2|
+--------------------+------------+------+------------+--------+-----+---------+--------+
|       Alphonse Paul|+33223235223|   214|          R1|     DOC|false|      EDP|      NA|
|         Ammari Zied|+33223235811|   209|          R1|      MC| true|      EDP|      NA|
|         André Simon|+33223237555|   301|          R1|     DOC|false| THEO-ERG|      NA|
|        Angst Jurgen|+33223236519|   320|          R1|      MC|false|PROC-STOC|      NA|
|     Bailleul Ismaël|+33223236369|   302|          R1|      MC| true| THEO-ERG|      NA|
|          Baker Mark|+33223236028|   835|          R1|      PR| true|      GAN|      NA|
|      Balac Stephane|+33223236274|   110|          R1|      MC|false|   ANANUM|      NA|
|           Bauer Max|+33223236675|   734|          R1|      MC|false|      GAN|      NA|
|     Bavard Juliette|+33223236724|   331|        CNRS|      CR|false|      GAN|THEO-ERG|
|    Beauchard Karine|+33223236164|   235|          R1|      PR| true|   ANANUM|      NA|
|        Bekka Bachir|+33223235779|   307|          R1|      PR| true| THEO-ERG|      NA|
|         Bekka Karim|+33223236180|   615|          R1|      MC|false|      G&S|      NA|
|      Belgacem Maher|+33223236670|    NA|         EXT|     DOC|false|   ANANUM|      NA|
|    Bellis Alexandre|+33223236696|   634|          R1|     DOC|false|      GAN|      NA|
|     Belmiloudi Aziz|+33223238646|    NA|        INSA|      MC| true|   ANANUM|      NA|
|     Ben Elouefi Rim|+33223236670|    NA|         EXT|     DOC|false|     STAT|      NA|
|   Benasseni Jacques|+33299141822|    NA|          R2|      PR| true|     STAT|      NA|
|Bennani-Dosse Moh…|+33299141796|    NA|          R2|      MC|false|     STAT|      NA|
|     Bernier Joachim|+33223237558|   214|          R1|     DOC|false|   ANANUM|      NA|
|    Berthelot Pierre|+33223236043|   601|          R1|      PE| true|       GA|      NA|
+--------------------+------------+------+------------+--------+-----+---------+--------+
only showing top 20 rows
```

17.6.1 Schema

In [7]: `df.printSchema()`

```
root
 |-- name: string (nullable = true)
 |-- phone: string (nullable = true)
 |-- office: string (nullable = true)
 |-- organization: string (nullable = true)
 |-- position: string (nullable = true)
 |-- hdr: boolean (nullable = true)
 |-- team1: string (nullable = true)
 |-- team2: string (nullable = true)
```

17.6.2 display

```
In [8]: display(df)
```

```
DataFrame[name: string, phone: string, office: string, organization: string, position: string, hdr: bool
```

17.6.3 select

```
In [9]: df.select(df["name"], df["position"], df["organization"])
```

```
Out[9]: DataFrame[name: string, position: string, organization: string]
```

```
In [10]: df.select(df["name"], df["position"], df["organization"]).show()
```

```
+--------------------+--------+------------+
|                name|position|organization|
+--------------------+--------+------------+
|       Alphonse Paul|     DOC|          R1|
|         Ammari Zied|      MC|          R1|
|         André Simon|     DOC|          R1|
|        Angst Jurgen|      MC|          R1|
|     Bailleul Ismaël|      MC|          R1|
|          Baker Mark|      PR|          R1|
|      Balac Stephane|      MC|          R1|
|           Bauer Max|      MC|          R1|
|     Bavard Juliette|      CR|        CNRS|
|    Beauchard Karine|      PR|          R1|
|        Bekka Bachir|      PR|          R1|
|         Bekka Karim|      MC|          R1|
|      Belgacem Maher|     DOC|         EXT|
|    Bellis Alexandre|     DOC|          R1|
|     Belmiloudi Aziz|      MC|        INSA|
|     Ben Elouefi Rim|     DOC|         EXT|
|   Benasseni Jacques|      PR|          R2|
|Bennani-Dosse Moh...|      MC|          R2|
|     Bernier Joachim|     DOC|          R1|
|    Berthelot Pierre|      PE|          R1|
+--------------------+--------+------------+
only showing top 20 rows
```

17.6.4 filter

```
In [11]: df.filter(df["organization"] == "R2").show()
```

```
+--------------------+------------+------+------------+--------+-----+---------+-----+
|                name|       phone|office|organization|position|  hdr|    team1|team2|
+--------------------+------------+------+------------+--------+-----+---------+-----+
|   Benasseni Jacques|+33299141822|    NA|          R2|      PR| true|     STAT|   NA|
|Bennani-Dosse Moh...|+33299141796|    NA|          R2|      MC|false|     STAT|   NA|
|Cornillon Pierre-...|+33299141819|    NA|          R2|      MC|false|     STAT|   NA|
|     Fromont Magalie|+33299053264|    NA|          R2|      PR| true|     STAT|   NA|
|Giacofci Joyce Ma...|+33299141800|    NA|          R2|      MC|false|     STAT|   NA|
|Klutchnikoff Nicolas|+33299141819|    NA|          R2|      MC|false|     STAT|   NA|
```

```
|     Le Guevel Ronan|+33299141800|    NA|           R2|      MC|false|PROC-STOC| STAT|
|           Mom Alain|+33299141808|    NA|           R2|      MC|false|     STAT|   NA|
|        Morvan Marie|+33223236670|    NA|           R2|     DOC|false|     STAT|   NA|
|     Pelletier Bruno|+33299141807|    NA|           R2|      PR| true|     STAT|   NA|
|    Rouviere Laurent|+33299141804|    NA|           R2|      MC|false|     STAT|   NA|
+--------------------+------------+------+-------------+--------+-----+---------+-----+
```

17.6.5 filter + select

```
In [12]: df2 = df.filter(df["organization"] == "R2").select(df['name'],df['team1'])
```

```
In [13]: df2.show()
```

```
+--------------------+---------+
|                name|    team1|
+--------------------+---------+
|   Benasseni Jacques|     STAT|
|Bennani-Dosse Moh...|     STAT|
|Cornillon Pierre-...|     STAT|
|     Fromont Magalie|     STAT|
|Giacofci Joyce Ma...|     STAT|
|Klutchnikoff Nicolas|     STAT|
|     Le Guevel Ronan|PROC-STOC|
|           Mom Alain|     STAT|
|        Morvan Marie|     STAT|
|     Pelletier Bruno|     STAT|
|    Rouviere Laurent|     STAT|
+--------------------+---------+
```

17.6.6 orderBy

```
In [14]: (df.filter(df["organization"] == "R2")
            .select(df["name"],df["position"])
            .orderBy("position")).show()
```

```
+--------------------+--------+
|                name|position|
+--------------------+--------+
|        Morvan Marie|     DOC|
|Cornillon Pierre-...|      MC|
|Bennani-Dosse Moh...|      MC|
|Giacofci Joyce Ma...|      MC|
|           Mom Alain|      MC|
|Klutchnikoff Nicolas|      MC|
|    Rouviere Laurent|      MC|
|     Le Guevel Ronan|      MC|
|   Benasseni Jacques|      PR|
|     Fromont Magalie|      PR|
|     Pelletier Bruno|      PR|
+--------------------+--------+
```

17.6.7 groupBy

```
In [15]: df.groupby(df["hdr"])

Out[15]: <pyspark.sql.group.GroupedData at 0x7fdb1e1b2730>

In [16]: df.groupby(df["hdr"]).count().show()

+-----+-----+
|  hdr|count|
+-----+-----+
| true|  103|
|false|  141|
+-----+-----+
```

WARNING: Don't confuse GroupedData.count() with DataFrame.count(). GroupedData.count() is not an action. DataFrame.count() is an action.

```
In [17]: df.filter(df["hdr"]).count()

Out[17]: 103

In [18]: df.filter(df['hdr']).select("name").show()

+--------------------+
|                name|
+--------------------+
|         Ammari Zied|
|     Bailleul Ismaël|
|          Baker Mark|
|    Beauchard Karine|
|        Bekka Bachir|
|     Belmiloudi Aziz|
|   Benasseni Jacques|
|    Berthelot Pierre|
|       Bourqui David|
|Breton Jean-Chris…|
|         Briane Marc|
|        Cadre Benoît|
|       Caloz Gabriel|
|        Cantat Serge|
|       Caruso Xavier|
|   Castella Francois|
|       Causeur David|
|   Cerveau Dominique|
|   Chartier Philippe|
|   Chauvet Guillaume|
+--------------------+
only showing top 20 rows

In [19]: df.groupBy(df["organization"]).count().show()
```

```
+------------+-----+
|organization|count|
+------------+-----+
|         ENS|    3|
|        CNRS|   19|
|        INSA|   19|
|          R2|   11|
|       INRIA|    9|
|        AGRO|    5|
|         EXT|    2|
|          R1|  176|
+------------+-----+
```

17.6.8 Exercises

- How many teachers from INSA (PR+MC) ?
- How many MC in STATS team ?
- How many MC+CR with HDR ?
- What is the ratio of student supervision (DOC / HDR) ?
- List number of people for every organization ?
- List number of HDR people for every team ?
- Which team contains most HDR ?
- List number of DOC students for every organization ?
- Which team contains most DOC ?
- List people from CNRS that are neither CR nor DR ?

```
In [20]: df.select("organization").filter(df["organization"]=="INSA").count()
```

```
Out[20]: 19
```

```
In [21]: (df.select(["position", "team1", "team2"])
          .filter((df["team1"]=="STAT") | (df["team2"]=="STAT"))
          .filter(df["position"] == "MC").count())
```

```
Out[21]: 15
```

```
In [22]: (df.select(["position", "hdr"])
          .filter((df["position"]=="MC") | (df["position"]=="CR"))
          .filter(df["hdr"]).count())
```

```
Out[22]: 28
```

```
In [23]: (df.select("position").filter(df["position"]=="DOC").count() /
          df.select(df["hdr"]).filter(df["hdr"]).count())
```

```
Out[23]: 0.6019417475728155
```

```
In [24]: (df.select(["hdr", "team1", "team2"])
          .filter("hdr")
          .rdd.flatMap(lambda row: (row.team1, row.team2))
          .filter(lambda v : v != 'NA')
          .map(lambda row : (row,1))
          .reduceByKey(lambda a, b:a+b)
          .sortBy(lambda v: -v[1])
          .collect()
         )
```

```
Out[24]: [('ANANUM', 21),
          ('THEO-ERG', 14),
          ('STAT', 14),
          ('EDP', 11),
          ('G&S', 9),
          ('GAN', 9),
          ('GA', 8),
          ('GAE', 8),
          ('PROC-STOC', 7),
          ('MECA', 6),
          ('IREM', 2),
          ('ADM', 1)]
```

```
In [25]: (df.select(["position", "team1", "team2"])
          .filter(df.position=="DOC")
          .rdd.flatMap(lambda row: [row.team1, row.team2])
          .filter(lambda v : v != 'NA')
          .map(lambda row : (row,1))
          .reduceByKey(lambda a, b:a+b)
          .sortBy(lambda v: -v[1])
          .collect()
         )
```

```
Out[25]: [('ANANUM', 14),
          ('STAT', 9),
          ('THEO-ERG', 8),
          ('GAN', 8),
          ('PROC-STOC', 8),
          ('EDP', 7),
          ('MECA', 4),
          ('GAE', 4),
          ('GA', 4),
          ('G&S', 2)]
```

```
In [26]: import pyspark.sql.functions as f

         df1 = (df.select(["position", "team1", "hdr"])
          .filter(df.hdr)
          .groupBy("team1")
          .agg(f.count("position").alias("count1"))
         )
```

```
In [27]: df2 = (df.select(["position", "team2", "hdr"])
          .filter(df.hdr)
          .filter(df.team2 != "NA")
          .groupBy("team2")
          .agg(f.count("team2").alias("count2"))
         )
```

```
In [28]: df3 = (df1.join(df2, df1.team1 == df2.team2, how="left")
          .na.fill(0)
          .drop("team2"))
```

```
In [29]: df3.withColumn("total", df3.count1+df3.count2).orderBy("total", ascending=False).show()
```

```
+---------+------+------+-----+
|    team1|count1|count2|total|
+---------+------+------+-----+
```

```
|   ANANUM|    21|     0|   21|
|     STAT|    14|     0|   14|
| THEO-ERG|    11|     3|   14|
|      EDP|    10|     1|   11|
|      GAN|     9|     0|    9|
|      G&S|     8|     1|    9|
|      GAE|     8|     0|    8|
|       GA|     7|     1|    8|
|PROC-STOC|     6|     1|    7|
|     MECA|     6|     0|    6|
|     IREM|     2|     0|    2|
|      ADM|     1|     0|    1|
+---------+------+------+-----+
```

```
In [30]: (df.filter((df.position=="DOC") & (df.team1 == "ANANUM"))
          .select("name")
          .show()
         )
```

```
+--------------------+
|                name|
+--------------------+
|      Belgacem Maher|
|     Bernier Joachim|
|       Calvez Adrien|
|        Corre Samuel|
|      Dao Manh Khang|
|       Doli Valentin|
|     Fontaine Marine|
|       Horsin Romain|
| Joannopoulos Emilie|
|     Le Balc'h Kevin|
|        Moitier Zoïs|
|Nguyen Thi-Hoai-T…|
|      Rosello Angelo|
|      Tusseau Maxime|
+--------------------+
```

```
In [31]: (df.select("organization")
          .groupby("organization").count().show())
```

```
+------------+-----+
|organization|count|
+------------+-----+
|         ENS|    3|
|        CNRS|   19|
|        INSA|   19|
|          R2|   11|
|       INRIA|    9|
|        AGRO|    5|
|         EXT|    2|
|          R1|  176|
```

```
+------------+-----+
```

```
In [32]: (df.select(["name","organization","position"])
          .filter((df.position == "DR") | (df.position == "CR"))
          .show())
```

```
+-------------------+------------+--------+
|               name|organization|position|
+-------------------+------------+--------+
|    Bavard Juliette|        CNRS|      CR|
|Bonthonneau Yannick|        CNRS|      CR|
|       Cantat Serge|        CNRS|      DR|
|      Caruso Xavier|        CNRS|      CR|
|     Cérou Frédéric|       INRIA|      CR|
|  Chartier Philippe|       INRIA|      DR|
|        Coulon Rémi|        CNRS|      CR|
|Crouseilles Nicolas|       INRIA|      CR|
|      Dauge Monique|        CNRS|      DR|
|    Duchene Vincent|        CNRS|      CR|
|     Erhel Jocelyne|       INRIA|      DR|
|         Faou Erwan|       INRIA|      DR|
|        Gros Michel|        CNRS|      CR|
|       Héas Patrick|        CNRS|      CR|
|      Herzet Cedric|        CNRS|      CR|
|    Kleptsyn Victor|        CNRS|      CR|
|  Le Gland François|       INRIA|      DR|
|     Lemou Mohammed|        CNRS|      DR|
|        Loray Frank|        CNRS|      DR|
|      Memin Etienne|       INRIA|      DR|
+-------------------+------------+--------+
only showing top 20 rows
```

```
In [33]: (df.select(["name","organization","position"])
          .filter(df.organization == "CNRS")
          .filter((df.position != "DR") & (df.position != "CR"))
          .groupBy("position").count().show())
```

```
+--------+-----+
|position|count|
+--------+-----+
|      TC|    2|
|      IR|    2|
|      AI|    1|
|      IE|    1|
+--------+-----+
```

```
In [34]: sc.stop()
```

Chapter 18

Dask dataframes on HDFS

To use Dask dataframes in parallel across an HDFS cluster to read CSV data. We can coordinate these computations with distributed and dask.dataframe.

As Spark, Dask can work in cluster mode. You can use the dask module dask_jobqueue to launch a Dask cluster with the job manager SLURM.

```
In [1]: from dask_jobqueue import SLURMCluster

        cluster = SLURMCluster(cores=16,
                               queue='test',
                               project='myproject',
                               memory="16GB",
                               walltime="01:00:00")
```

The cluster generates a traditional job script and submits that an appropriate number of times to the job queue. You can see the job script that it will generate as follows:

```
In [2]: print(cluster.job_script())

#!/usr/bin/env bash

#SBATCH -J dask-worker
#SBATCH -p test
#SBATCH -A myproject
#SBATCH -n 1
#SBATCH --cpus-per-task=16
#SBATCH --mem=15G
#SBATCH -t 01:00:00

/usr/share/miniconda3/envs/big-data/bin/python -m distributed.cli.dask_worker tcp://10.1.0.4:40489 --ntl
```

Access to the cluster using following lines:

```
import dask.dataframe as dd
from dask.distributed import Client
client = Client(cluster)
```

`nyc2014` is a dask.dataframe objects which present a subset of the Pandas API to the user, but farm out all of the work to the many Pandas dataframes they control across the network.

```
nyc2014 = dd.read_csv('/opt/datasets/nyc-data/2014/yellow*.csv',
parse_dates=['pickup_datetime', 'dropoff_datetime'],
skipinitialspace=True)
nyc2014 = c.persist(nyc2014)
progress(nyc2014)
```

18.0.1 Exercises

- Display head of the dataframe
- Display number of rows of this dataframe.
- Compute the total number of passengers.
- Count occurrences in the payment_type column both for the full dataset, and filtered by zero tip (tip_amount == 0).
- Create a new column, tip_fraction
- Plot the average of the new column tip_fraction grouped by day of week.
- Plot the average of the new column tip_fraction grouped by hour of day.

Dask dataframe documentation

```
In [3]: # import dask.dataframe as dd
        # from distributed import Client, progress
        #
        # c = Client('127.0.0.1:8786')
        # nyc2014 = dd.read_csv('hdfs://svmass2.mass.uhb.fr:54310/user/datasets/nyc-tlc/2014/yellow*.csv',
        # parse_dates=['pickup_datetime', 'dropoff_datetime'],
        # skipinitialspace=True)
        #
        # nyc2015 = dd.read_csv('hdfs://svmass2.mass.uhb.fr:54310/user/datasets/nyc-tlc/2015/yellow*.csv',
        # parse_dates=['tpep_pickup_datetime', 'tpep_dropoff_datetime'])
        # nyc2014, nyc2015 = c.persist([nyc2014, nyc2015])
        #
        # progress(nyc2014, nyc2015)
```

Chapter 19

Spark dataframes on HDFS

New York City Taxi Cab Trip

We look at the New York City Taxi Cab dataset. This includes every ride made in the city of New York since 2009.

On this website you can see the data for one random NYC yellow taxi on a single day.

On this post, you can see an analysis of this dataset. Postgres and R scripts are available on GitHub.

19.1 Loading the data

Normally we would read and load this data into memory as a Pandas dataframe. However in this case that would be unwise because this data is too large to fit in RAM.

The data can stay in the hdfs filesystem but for performance reason we can't use the csv format. The file is large (32Go) and text formatted. Data Access is very slow.

You can convert csv file to parquet with Spark.

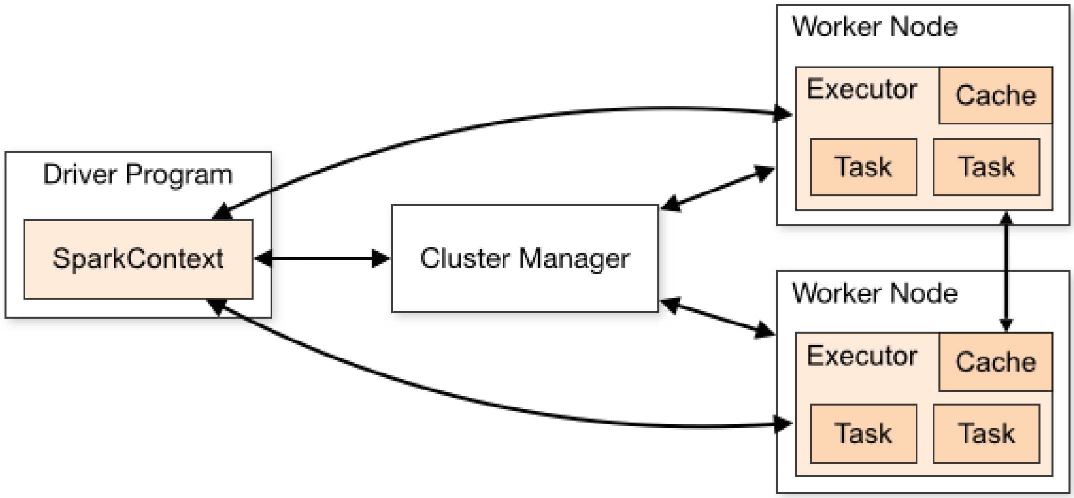
```
from pyspark.sql import SparkSession
spark = SparkSession.builder \
        .master("spark://svmass2.mass.uhb.fr:7077") \
        .config('spark.hadoop.parquet.enable.summary-metadata', 'true') \
        .config("spark.cores.max", "10") \
        .getOrCreate()
df = spark.read.csv(
"hdfs://svmass2.mass.uhb.fr:54310/user/datasets/nyc-tlc/2009/yellow_tripdata_2009-01.csv",
                    header="true",inferSchema="true")
df.write.parquet("hdfs://svmass2.mass.uhb.fr:54310/user/navaro_p/nyc-taxi/2019-01.parquet")
spark.stop()
```

To read multiple files

```
spark.read.format("csv").option("header", "true").load("../Downloads/*.csv")
```

19.2 Spark Cluster

A Spark cluster is available and described on this web interface




```
from pyspark.sql import SparkSession
spark = SparkSession.builder \
        .master('spark://svmass2.mass.uhb.fr:7077') \
        .getOrCreate()
spark
```

The SparkSession is connected to the Spark's own standalone cluster manager (It is also possible to use YARN). The manager allocate resources across applications. Once connected, Spark acquires executors on nodes in the cluster, which are processes that run computations and store data for your application. Next, it sends your application code (Python file) to the executors. Finally, tasks are sent to the executors to run.

Spark can access to files located on hdfs and it is also possible to access to local files. Example:

```
df = spark.read.parquet('file:///home/navaro_p/nyc-taxi/2016.parquet')
```

19.2.1 Exercise

- Pick a year and read and convert csv files to parquet in your hdfs homedirectory.
- **Don't run the python code inside a notebook cell**. Save a python script and launch it from a terminal instead. In Jupyter notebook you won't see any progress or information if error occurs.
- Use the `spark-submit` command shell to run your script on the cluster.
- You can control the log with

  ```
  spark.sparkContext.setLogLevel('ERROR')
  ```

  Valid log levels include: ALL, DEBUG, ERROR, FATAL, INFO, OFF, TRACE, WARN

Try your script with a single file before to do it for a whole year.

Read carefully the script given above, don't submit it as is. You have to change some part of this code

19.3 Some examples that can be run on the cluster

- Here we read the NYC taxi data files of year 2016 and select some variables.

```
columns = ['tpep_pickup_datetime', 'passenger_count', 'pickup_longitude', 'pickup_latitude', 'dropoff_lo

df = (spark.read.parquet('hdfs://svmass2.mass.uhb.fr:54310/user/navaro/nyc-taxi/2016.parquet').select(*c
```

- Sum the total number of passengers

  ```
  df.agg({'passenger_count': 'sum'}).collect()
  ```

- Average number of passenger per trip‘

  ```
  df.agg({'passenger_count': 'avg'}).collect()
  ```

- How many trip with 0,1,2,3,...,9 passenger‘

  ```
  df.groupby('passenger_count').agg({'*': 'count'}).collect()
  ```

19.4 Exercise

How well people tip based on the number of passengers in a cab. To do this you have to:

1. Remove rides with zero fare
2. Add a new column `tip_fraction` that is equal to the ratio of the tip to the fare
3. Group by the `passenger_count` column and take the mean of the `tip_fraction` column.

19.4.1 Cheat Sheets and documentation

- Spark DataFrames in Python
- Spark in Python
- https://spark.apache.org/docs/latest/api/python/pyspark.sql.html

Use the PySpark API.

- **Write a python program and use `spark-submit`**
- **Read the parquet files instead of csv files**
- **Don't forget spark.stop() at the end of the script**

19.5 Hints

- How to remove rows

  ```
  df = df.filter(df.name == 'expression')
  ```

- How to make new columns

  ```
  df = df.withColumn('var2', df.var0 + df.var1)
  ```

- How to do groupby-aggregations

  ```
  df.groupBy(df.name).agg({'column-name': 'avg'})
  ```

When you want to collect the result of your computation, finish with the `.collect()` method.

19.5.1 Exercices

1. Plot the tip as a function of the hour of day and the day of the week?
2. Investigate the `payment_type` column. See how well each of the payment types correlate with the `tip_fraction`. Did you find anything interesting? Any guesses on what the different payment types might be? If you're interested you may be able to find more information on the NYC TLC's website
3. How quickly can you get a taxi cab for a particular day of the year? How about for a particular hour of that day?