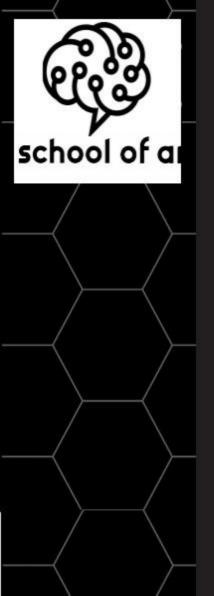


Data manipulation:
Data Files, and
Data Cleaning & Preparation

**AAA-Python Edition** 



#### Plan

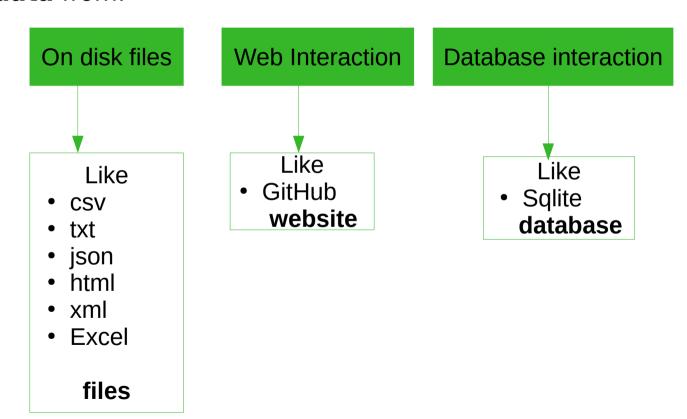
- 1- Data Files: Reading and Writing
- 2- Missing data
- 3- Data transformation
- 4- String Manipulation



L- Data Filles: Reading and Writin

#### pandas

 Using pandas, we can easily read (and write) different types of data from:





L- Data Files: Reading and Writi

#### On disk Files

You have just to choose the right function to use with the right

arguments:

No need to specify a header or a separator

1 #viewing the content of A3P-w2-ex1.csv
2 !cat A3P-w2-ex1.csv

**0** a

**3** 9 10 11

a,b,c,d,message 1,2,3,4,hello 5,6,7,8,world 9,10,11,12,foo The file has a **header**It is a **csv** file
It is delimited with ','

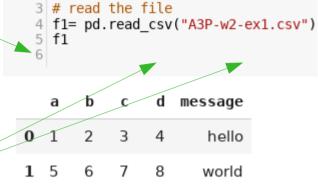
3 f2= pd.read\_csv("A3P-w2-ex1.csv", header=None sep=","

message

hello

world

foo



foo

In this case, no need

to specify a separator

Specifying that the data file has no header, a default Header was added

9 10

The real header is considered as a row value

In the case where
the delimiter is **not**a ',', you can
specify the used one
(you can also use
a regular expression
like: '\s+'==one ore

more spaces )
[By Amina Delali]



Write

```
On disk Files
```

```
2 !cat A3P-w2-ex2.csv
```

```
Specifying an index
  column: the fifth column
is no longer a value column
   but an index column
```

```
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
```

```
2>header=[ "col"+str(i) for i in range(1,6)]
```

3 ind= header[len(header)-1]

4 f3= pd.read\_csv("A3P-w2-ex2.csv", names=header,index\_col=ind) 5 **f3** 

Specifying a header: list of **5** values

Some files may contain rows values + **other text**, so you can **skip** this text by: skiprows argument: skiprows=[0, 2]: will not include the **first** and **third** rows

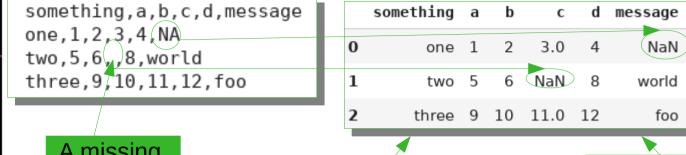
col5 hello world 9 10 foo 12

col1 col2 col3 col4

[By Amina Delali]



1- Data Files: Reading and Writing On disk Files
The content of the file: A3P-w2-ex5.csv
something,a,b,c,d,message something a b



A missing value

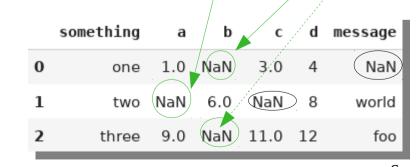
1 pd.read\_csv("A3P-w2-ex5.csv")

By default, the **missing** and **Na** values are considered to be **NULL** 

2 pd.read\_csv("A3P-w2-ex5.csv",na\_values={"a":,5,"b":[2,10]})

We can specify the **Null** values as a dictionary, to specify the **corresponding columns** as keys

We can also use a list, to select from all the values of the file





# S: A Writi

```
On disk Files
```

```
1 # we will read only 10 rows, sepearated in 5 chunks of 2 rows
 tfr=pd.read csv("A3P-w2-ex6.csv",nrows =10, chunksize=2)
                                                   1 # the initial file: number of rows = 10000
 for i, chunk in zip(range(1,6),tfr):
                                                   2 df=pd.read csv("A3P-w2-ex6.csv")
   print("chunk"+str(i)+":\n", chunk)
                                                   3 df.shape
               Combine the arguments
                                                 (10000, 5)
               values to create tuples
   chunk1:
                                         four key
                      two
                              three
                                                          Chunksize==
    0.467976 -0.038649 -0.295344 -1.824726
   1 -0.358893 1.404453 0.704965 -0.200638
                                               В
                                                             2 rows
   chunk2:
                                                           We read only
                                         four key
            one
                      two
                              three
   2 -0.501840  0.659254 -0.421691 -0.057688
                                                      10 rows (from 10000)
```

four key

four key

Total of **5**- chunks (2 \* 5== 10 rows )

```
6 -0.776764 0.935518 -0.332872 -1.875641 U
7 -0.913135 1.530624 -0.572657 0.477252 K
chunk5:

one two three four key
8 0.358480 -0.497572 -0.367016 0.507702 S
9 -1.740877 -1.160417 -1.637830 2.172201 G
```

3 0.204886 1.074134 1.388361 -0.982404

two

two

three

three

0.283763 -0.837063

0.419395 -2.251035

chunk3:

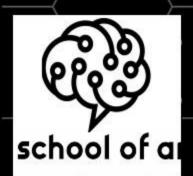
chunk4:

one

4 0.354628 -0.133116

5 1.817480 0.742273

one

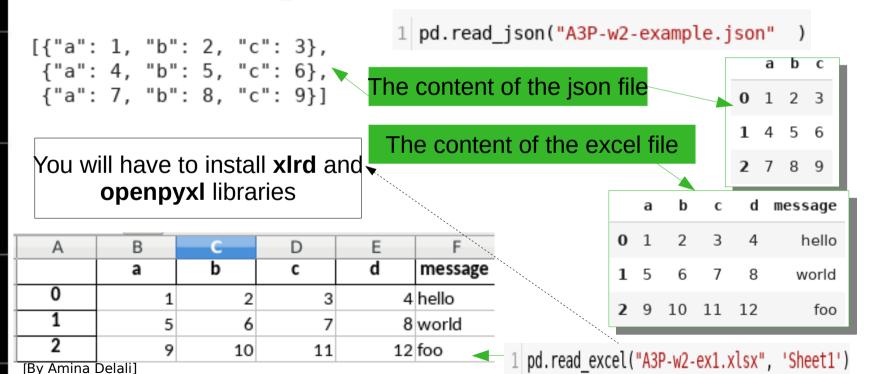


1- Data Files: Reading and Writing

#### On disk Files

With **read\_csv** or **read\_table**, you can read other text files format as (.txt files) containing columns separated by **delimiters**.

- You can use read\_json to read json files
- You can use read html to read tabular data in a html file.
- You can use read\_excel to read excel files.





es: nd

On disk Files

Will read all the tables

dfs= pd.read html("A3P-w2-fdic failed bank list.html") print("number of tables ==", len(dfs)) only the 5 rows will be displayed by default pd.options.display.max rows = 5 dfs[0] number of tables == 1

Only the **displayed** number of rows will be **limited** to **5** (the DataFrame still contain **all** the rows)

Acquiring Bank Name City ST Closing Date Updated Date CERT Institution September 23, November 17, 0 Allied Bank Mulberry AR 91 Todav's Bank 2016 2016

The required libraires (in addition to pandas) are: lxml, beautifulsoup4, and html5lib.

Preview of the html table

Bank Name	City	ST CERT Acquiring Institution	Closing Date	Updated Date
Allied Bank	Mulberry	AR 91 Today's Bank	September 23, 2016	November 17, 2016
The Woodbury Banking Company	Woodbury	GA 11297 United Bank	August 19, 2016	November 17, 2016 9

[By Amina Delali]

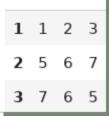


Write

 To write the data to a file, you can use this corresponding methods: to csv, to json, and to excel.

#### Creating a DataFrame

df = pd.DataFrame([range(1,4),range(5,8),range(7,4,-1)],index=range(1,4),columns=list("abc"))
df



#### Saving the files to different files format

Content of file1.txt and file1.csv

```
,a,b,c
1,1,2,3
2,5,6,7
3,7,6,5
```

```
df.to_csv("file1.csv")
df.to_csv("file1.txt")
df.to_json("file1.json")
df.to_excel("file1.xlsx")
```

#### Content of file1.json

#### Downloading file1.xlsx using this commands:

Content of file1.xlsx

	Α	В	С	D
		a	b	С
	1	1	2	3
1	2	5	6	7
	3	7	6	5

2 from google.colab import files
3 files.download('file1.xlsx')



Data Files:
 Reading and Writing

#### Web Interaction

It is possible to interact with websites APIs to retrieve data via a predefined format.

By default, it will get

```
only the last 30 issues
1 import requests
2 # url to get the first page of 30 issues of a GitHub Repository
3 urll = "https://api.github.com/repos/pandas-dev/pandas/issues"
4 # url to get the second page of 100 closed issues of a GitHub Repository,
 url2 = "https://api.github.com/repos/pandas-dev/pandas/issues?state=closed&page=2&per page=100"
                                            We selected only
 alliss= requests.get(url1)
9 closed= requests.get(url2)
                                       closed issues, the second
                                        page, and each page will
  8 alliss= requests.get(url1)
                                           contain 100 issues
   closed= requests.get(url2)
   # create DataFrames from the responses of the request
    da= alliss.json()
    daf=pd.DataFrame(da,columns=["id","state"]) We selected from the json
   # create DataFrames from
                                                     data this 2 columns
 15 dc= closed.json()
    dcf=pd.DataFrame(dc,columns=["id","state"])
   print(daf.shape)
 19 print(dcf.shape)
                                    The two columns
```

we selected



1- Data Files: Reading and Writing

#### **DataBase Interaction**

True

[By Amina Delali]

- In the following example, we will use sqlachemy and pandas to interact with an sqlite database.
- There is various ways to connect, create and extract data from a DataBase using sqlalchemy. We selected one of them.

```
import sqlalchemy as sqla
from sqlalchemy import Column, Table, types, MetaData
# if the database example1.db doesn't exist , the following statement will create it
DB= sqla.create_engine("sqlite:///A3P-w2-example1.db")
                                                                The name of
                                                                the database
# to use "meta" later for the creation of the Table
meta = MetaData(DB)
                                  Link "meta" with the
                               created database (engine)
# define a table's scheme
myTable=Table("myTab")meta_Column("id", types.String),Column("value",types.Integer))
# creation of the Table
                               The table is linked
myTable.create()
                                    with DB
                                                       The table will have
# verification if the table exist
                                                    2 columns: id and value
DB.has table("myTab")
                           The name of the table
                                                                             12
```

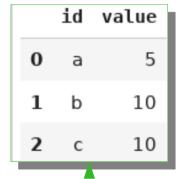


Writi

#### DataBase Interaction

```
2 # verify if the corresponding file is created
3 !ls
 # specify the values to insert into the created table
6 insertion= myTable.insert().values([{"id": "a", "value":5},{"id": "b", "value":10},{"id": "c",
7 # insert the values nto the table
8 DB.execute(insertion)
                          The created DataBase
     A3P-w2-ex5.csv
     A3P-w2-ex6.csv
     A3P-w2-ex7.csv
     A3P-w2-example1.db
     A3P-w2-example.json
     A3P-w2-fdic failed bank list.html
  1 # read the content of the table into a dataframe
  2 pd.read sql("myTab",DB)
```

Value to insert with the corresponding column

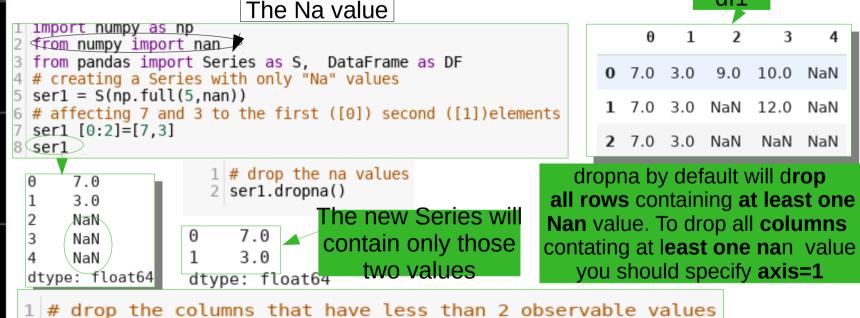




## 2- Missing data

#### Filtering out

• Sometimes, data may have **missing** or "**Na**" values. So, with **pandas** we can **filter out** those values using the **dropna** method.



1 # drop the columns that have less than 2 observable value 2 df1.dropna(axis=1,thresh=2)

Column 3 is kept, because it has two values different from Nan .

[By Amina Delali]

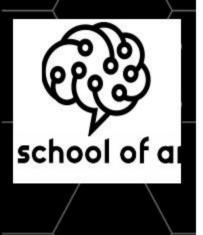
o 7.0 3.0 10.0

1 7.0 3.0 12.0

1 7.0 3.0 NaN

2 df1.dropna(how="all")

how="all" means that dropna will drop rows if all the values are "Na"



#### data 9 Missin 2

#### Filling in

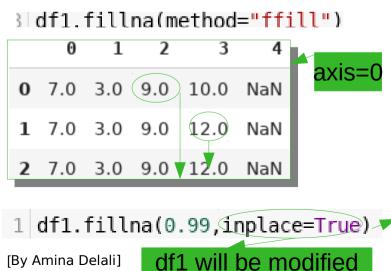
7.0 3.0 0.0

**Instead** of dropping missing data, we can produce **new ones** using



By default, **fillna** will fill rows (axis=0) with: - a given **value**: in this case **limit=2** signify the maximum number of nan values to be replaced in each **column (this is our case)** 

- a given **method**:in this case **limit=2** signify the maximum number of **consecutive** Nan values to be replaced in a column



0.0

NaN

If axis=1 was specified 7.0 3.0 9.0 10.0 10.0 7.0 (3.0) 3.0 (12.0) 12.0 3.0/3.03.0 3.0 7.0 3.0 9.00 10.00 0.99 3.0 0.99 12.00 0.99

0.99 0.99

7.0 3.0 0.99

15

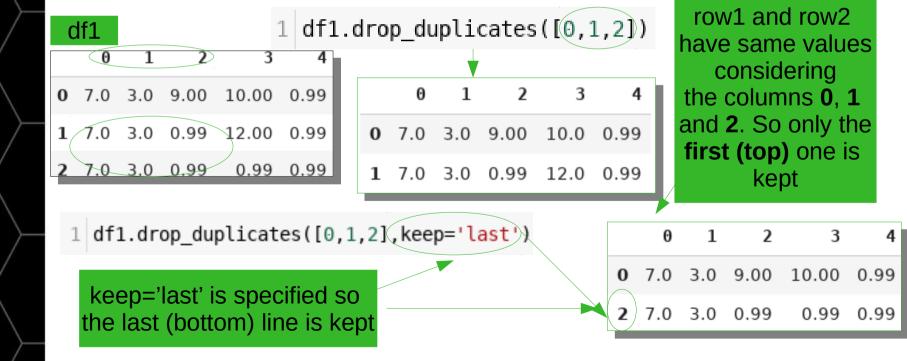


**Data Transformation** 

3-

 Some other types of transformations are necessary as: dropping duplicated data, transforming and creating new data using mapping, replacing values, renaming indexes, discretization, permutation and random sampling

Dropping duplicates





### **Transformation** Data 3-

#### Transforming data

```
# indexing a column that does not exist, will create it
df1["chars"]=list("abc")
df1
```

	0	1	2	3	4	chars	order
0	7.0	3.0	9.00	10.00	0.99	a	First letter
1	7.0	3.0	0.99	12.00	0.99	b	Second letter
2	7.0	3.0	0.99	0.99	0.99	С	Third letter

chars 7.0 3.0 9.00 10.00 0.99 а 7.0 3.0 0.99 12.00 0.99 b **2** 7.0 3.0 0.99 0.99 0.99

Added the new column "order", by mapping the values from "chars" using the dictionary myMap

```
1 # create a mapping using a dict
2 myMap ={ "a":"First letter","c":"Third letter","b":"Second letter"}
3 # create a new column using that mapping
```

4 df1["order"]=df1["chars"].map(myMap)

1 # transform ther order column values to uppercase 2 df1["order"]=df1["order"].str.upper()

order FIRST LETTER SECOND LETTER THIRD LETTER

17



3- Data Transformation

#### Replacing values and Renaming indexes

```
1 # replacing 0.99 values by 1 and 12 by 120
2 df2=df1.replace([0.99,12],[1.120]) df2
```

- to modify only one value: df1.replace(0.99,1)
- using inplace=True, will modify the original DataFrame
- 0
   7.0
   3.0
   9.0
   10.0
   1.0
   a
   FIRST LETTER

   1
   7.0
   3.0
   1.0
   120.0
   1.0
   b
   SECOND LETTER

   2
   7.0
   3.0
   1.0
   1.0
   c
   THIRD LETTER
- 1 # replacing 0.99 values by 1 and 12 by 120 using a dictionary 2 df2=df1.replace({0.99:1,12:120})

3 doesn't exist

chars

2 df2=df1.rename(index={0:"zero",1:"one",3:"three"})

df2

order

- To modify columns

   labels use: column=
   if indexes or columns
   were strings we could
   use for example:
   index= str.lower()
- vero
   7.0
   3.0
   9.00
   10.00
   0.99
   a
   FIRST LETTER

   one
   7.0
   3.0
   0.99
   12.00
   0.99
   b
   SECOND LETTER

   2
   7.0
   3.0
   0.99
   0.99
   0.99
   c
   THIRD LETTER

[By Amina Delali]



### Transformation Data 3-

```
Discretization
```

[pd.cut(ser2,[0,4,7,9])

```
ser2
                                 NaN
          0 doesn't
                                         The values are grouped in 3
                               (0, 4]
            Belong
                              (0, 4]
                                         categories: 0 \rightarrow 4, 5 \rightarrow 7, 8 \rightarrow 9
                              (0, 4]
            to Any
                              (0, 4]
           category
                                         (0,4],(4,7],(7,9]
                              (4, 7]
                              (4, 7]
                                         - "(" means the value is out. The
                              (4, 7]
                                         "]" means the value is in.
                              (7.91
                         dtype: category
                         Categories (3, interval[int64]): [(0, 4] < (4, 7] < (7, 9]]
 dtype: int64
```

# grouping the ser2 values into four categories with the same length
pd.cut(ser2,4)

```
0 (-0.008, 2.0]

1 (-0.008, 2.0]

2 (-0.008, 2.0]

3 (2.0, 4.0]

4 (2.0, 4.0]

5 (4.0, 6.0]

6 (4.0, 6.0]

7 (6.0, 8.0]

8 (6.0, 8.0]
```

dtype: category

The values are grouped in 4 categories with the same length using the minimum and maximum values. All the values are included.

Categories (4, interval[float64]): [(-0.008, 2.0] < (2.0, 4.0] < (4.0, 6.0] < (6.0, 8.0]]



3- Data Transformation

#### Permutation

- 1 # create an array with not ordred range values (permuted)
- 2 norder= np.random.permutation(3)
- 1 # creating a new df reordred 2 df2.take(norder)

The **length** of the **array**must be **==** to the **number** of **rows** 

 Zero
 7.0
 3.0
 9.00
 10.00
 0.99
 a
 FIRST LETTER

 2
 7.0
 3.0
 0.99
 0.99
 0.99
 c
 THIRD LETTER

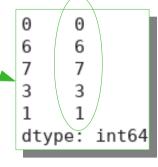
 one
 7.0
 3.0
 0.99
 12.00
 0.99
 b
 SECOND LETTER

Random sampling

1 # selecting randomly 5 values from ser2

2 ser2.sample(n=5)

If you select **n greater** than **ser2 length** you will have to specify: **replace=True** argument (to fill the reaming needed values)



array([0, 2, 1])



## 4- String manipulation

#### String methods

String object have useful methods that can be used:

```
1 # split a string specifying a separator
2 myStr= "This is an example"
                                                 ['This', 'is', 'an', 'example']
3 splitted=mvStr.split(" ")
   1 # join strings with a separator
                                                 'This-is-an-example'
   2 "-".join(splitted)
 1 # replace a value in string by another value
                                                      'This is an example'
 2 newStr=myStr.replace(" "," ")
 1 # find a value in a string using find
 2 myStr.find("e")
                                           1 # find a value in a string using index
                                          2 myStr.index("e")
 If "e" doesn't exist it
      will return -1
                                           If "e" doesn't exist it
                                         will raise an exception
1 # find a value in a string using in
2 "e" in myStr
               1 # the number of substring in a string
                                                                               21
               2 myStr.count(" ")
[By Amina Delali]
```



#### References

- SQLAlchemy authors and contributors. Sqlalchemy 1.2 documentation. On-line at https://docs.sqlalchemy.org/en/latest/core/dml.html. Ac-cessed on 19-10-2018.
- [2] GitHub. Rest api v3. On-line at https://developer.github.com/v3/.
   Accessed on 15-10-2018.
- [3] Wes McKinney. Python for data analysis: Data wrangling with Pandas, NumPy, and IPython. O'Reilly Media, Inc, 2018.
- [4] pydata.org. Pandas documentation. On-line at https://pandas.pydata.org/. Accessed on 19-10-2018.
- [5] pysheeet. Python sqlalchemy cheatsheet. On-line at https://pysheeet.readthedocs.io/en/latest/notes/python-sqlalchemy.html. Accessedon 19-10-2018.
- [6] McKinney Wes. pydata-book. On-line at https://github.com/wesm/pydata-book.git. Accessed on 14-10-2018.



### Thank you!

FOR ALL YOUR TIME