



Data manipulation: Data Files, and Data Cleaning & Preparation

AAA-Python Edition



school of analytics

Plan

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



school of ai

1- Data Files: Reading and Writing

pandas

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

On disk files



- Like
- csv
 - txt
 - json
 - html
 - xml
 - Excel

files

Web Interaction



- Like
- GitHub
website

Database interaction



- Like
- Sqlite
database



school of a

1- Data Files: Reading and Writing

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
```

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=",")
4 f2
```

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]

0	1	2	3	4	
0	a	b	c	d	message
1	1	2	3	4	hello
2	5	6	7	8	world
3	9	10	11	12	foo

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

The real header is
considered as a row value

```
3 # read the file
4 f1= pd.read_csv("A3P-w2-ex1.csv")
5 f1
6
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

In this case, no need
to specify a separator



school of a

1- Data Files: Reading and Writing

On disk Files

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

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

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

```
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

col1 col2 col3 col4

col5

	col1	col2	col3	col4
hello	1	2	3	4
world	5	6	7	8
foo	9	10	11	12

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



school of a

1- Data Files: Reading and Writing

On disk Files

The content of the
file : A3P-w2-ex5.csv

```
something,a,b,c,d,message  
one,1,2,3,4,NA  
two,5,6,8,world  
three,9,10,11,12,foo
```

A missing
value

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

```
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

	something	a	b	c	d	message
0	one	1.0	NaN	3.0	4	NaN
1	two	NaN	6.0	NaN	8	world
2	three	9.0	NaN	11.0	12	foo



school of ai

1- Data Files: Reading and Writing

On disk Files

```
1 # we will read only 10 rows, sepearated in 5 chunks of 2 rows
2 tfr=pd.read_csv("A3P-w2-ex6.csv",nrows =10, chunksize=2)
3
4 for i, chunk in zip(range(1,6),tfr):
5     print("chunk"+str(i)+"\n", chunk)
```

Combine the arguments
values to create tuples

chunk1:

	one	two	three	four	key
0	0.467976	-0.038649	-0.295344	-1.824726	L
1	-0.358893	1.404453	0.704965	-0.200638	B

chunk2:

	one	two	three	four	key
2	-0.501840	0.659254	-0.421691	-0.057688	G
3	0.204886	1.074134	1.388361	-0.982404	R

chunk3:

	one	two	three	four	key
4	0.354628	-0.133116	0.283763	-0.837063	Q
5	1.817480	0.742273	0.419395	-2.251035	Q

chunk4:

	one	two	three	four	key
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

```
1 # the initial file: number of rows = 10000
2 df=pd.read_csv("A3P-w2-ex6.csv")
3 df.shape
```

(10000, 5)

Chunksize==
2 rows

We read only
10 rows (from 10000)

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



school of ai

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.

```
[{"a": 1, "b": 2, "c": 3},  
 {"a": 4, "b": 5, "c": 6},  
 {"a": 7, "b": 8, "c": 9}]
```

```
1 pd.read_json("A3P-w2-example.json" )
```

The content of the json file

	a	b	c
0	1	2	3
1	4	5	6
2	7	8	9

You will have to install **xlrd** and **openpyxl** libraries

The content of the excel file

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

A	B	C	D	E	F
	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

```
1 pd.read_excel("A3P-w2-ex1.xlsx", 'Sheet1')
```

[By Amina Delali]



school of ai

1- Data Files: Reading and Writing

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]
```

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

number of tables == 1

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



school of ai

1- Data Files: Reading and Writing

- To **write** the data to a file, you can use this **corresponding methods**: `to_csv`, `to_json`, and `to_excel`.

Creating a DataFrame

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

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

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
```

```
1 df.to_csv("file1.csv")
2 df.to_csv("file1.txt")
3 df.to_json("file1.json")
4 df.to_excel("file1.xlsx")
```

Content of file1.json

```
{"a":{"1":1,"2":5,"3":7},"b":{"1":2,"2":6,"3":6},"c":{"1":3,"2":7,"3":5}}
```

Downloading file1.xlsx using
this commands:

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

Content of file1.xlsx

A	B	C	D
	a	b	c
1	1	2	3
2	5	6	7
3	7	6	5



school of ai

1- Data Files: Reading and Writing

Web Interaction

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

```
1 import requests
2 # url to get the first page of 30 issues of a GitHub Repository
3 url1 = "https://api.github.com/repos/pandas-dev/pandas/issues"
4 # url to get the second page of 100 closed issues of a GitHub Repository,
5 url2 = "https://api.github.com/repos/pandas-dev/pandas/issues?state=closed&page=2&per_page=100"
6
7
8 alliss= requests.get(url1)
9 closed= requests.get(url2)
```

By default, it will get only the last 30 issues

We selected only **closed** issues, the **second** page, and each page will contain **100** issues

```
8 alliss= requests.get(url1)
9 closed= requests.get(url2)
10
11 # create DataFrames from the responses of the request
12 da= alliss.json()
13 daf=pd.DataFrame(da,columns=["id","state"])
14 # create DataFrames from
15 dc= closed.json()
16 dcf=pd.DataFrame(dc,columns=["id","state"])
17
18 print(daf.shape)
19 print(dcf.shape)
```

We selected from the **json** data this **2** columns

(100, 2)
(30, 2)

The two columns we selected



school of a

1- Data Files: Reading and Writing

DataBase Interaction

- In the following example, we will use **sqlalchemy** 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**.

```
1 import sqlalchemy as sqla
2 from sqlalchemy import Column, Table, types, MetaData
3
4 # if the database example1.db doesn't exist , the following statement will create it
5 DB= sqla.create_engine("sqlite:///A3P-w2-example1.db")
6
7 # to use "meta" later for the creation of the Table
8 meta = MetaData(DB)
9
10 # define a table's scheme
11 myTable=Table("myTab",meta,Column("id",types.String),Column("value",types.Integer))
12 # creation of the table
13 myTable.create()
14 # verification if the table exist
15 DB.has_table("myTab")
```

The name of the database

Link "meta" with the created database (engine)

The table is linked with DB

The table will have 2 columns: id and value

The name of the table

True

[By Amina Delali]



school of ai

1- Data Files: Reading and Writing

DataBase Interaction

```
2 # verify if the corresponding file is created
3 !ls
4 # specify the values to insert into the created table
5
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
```

Value to insert with
the corresponding column

```
1 # read the content of the table into a dataframe
2 pd.read_sql("myTab",DB)
```

	id	value
0	a	5
1	b	10
2	c	10



school of ai

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.

The Na value

```
1 import numpy as np
2 from numpy import nan
3 from pandas import Series as S, DataFrame as DF
4 # creating a Series with only "Na" values
5 ser1 = S(np.full(5, nan))
6 # affecting 7 and 3 to the first ([0]) second ([1]) elements
7 ser1[0:2]=[7,3]
8 ser1
```

```
0    7.0
1    3.0
2    NaN
3    NaN
4    NaN
dtype: float64
```

```
1 # drop the na values
2 ser1.dropna()
```

```
0    7.0
1    3.0
dtype: float64
```

The new Series will contain only those two values

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

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

df1

	0	1	2	3	4
0	7.0	3.0	9.0	10.0	NaN
1	7.0	3.0	NaN	12.0	NaN
2	7.0	3.0	NaN	NaN	NaN

dropna by default will drop all rows containing at least one Nan value. To drop all columns containing at least one nan value you should specify **axis=1**

	0	1	3
0	7.0	3.0	10.0
1	7.0	3.0	12.0
2	7.0	3.0	NaN

```
2 df1.dropna(how="all")
```

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



school of ai

2- Missing data

Filling in

- Instead of dropping missing data, we can produce **new ones** using **pandas** with **fillna** method.

```
1 df1.fillna(0, limit=2)
```

	0	1	2	3	4
0	7.0	3.0	9.0	10.0	0.0
1	7.0	3.0	0.0	12.0	0.0
2	7.0	3.0	0.0	0.0	NaN

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**

```
3 df1.fillna(method="ffill")
```

	0	1	2	3	4
0	7.0	3.0	9.0	10.0	NaN
1	7.0	3.0	9.0	12.0	NaN
2	7.0	3.0	9.0	12.0	NaN

axis=0

```
1 df1.fillna(0.99, inplace=True)
```

df1 will be modified

If axis=1 was specified

	0	1	2	3	4
0	7.0	3.0	9.0	10.0	10.0
1	7.0	3.0	3.0	12.0	12.0
2	7.0	3.0	3.0	3.0	3.0
0	7.0	3.0	9.00	10.00	0.99
1	7.0	3.0	0.99	12.00	0.99
2	7.0	3.0	0.99	0.99	0.99



school of a

3- Data Transformation

- 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

df1

	0	1	2	3	4
0	7.0	3.0	9.00	10.00	0.99
1	7.0	3.0	0.99	12.00	0.99
2	7.0	3.0	0.99	0.99	0.99

```
1 df1.drop_duplicates([0,1,2])
```

	0	1	2	3	4
0	7.0	3.0	9.00	10.0	0.99
1	7.0	3.0	0.99	12.0	0.99

row1 and row2 have same values considering the columns 0, 1 and 2. So only the **first (top)** one is kept

```
1 df1.drop_duplicates([0,1,2], keep='last')
```

keep='last' is specified so the last (bottom) line is kept

	0	1	2	3	4
0	7.0	3.0	9.00	10.00	0.99
2	7.0	3.0	0.99	0.99	0.99



school of ai

3- Data Transformation

Transforming data

```
1 # indexing a column that does not exist, will create it
2 df1["chars"]=list("abc")
3 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	c	Third letter

	0	1	2	3	4	chars
0	7.0	3.0	9.00	10.00	0.99	a
1	7.0	3.0	0.99	12.00	0.99	b
2	7.0	3.0	0.99	0.99	0.99	c

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 the order column values to uppercase
2 df1["order"]=df1["order"].str.upper()
```

order
FIRST LETTER
SECOND LETTER
THIRD LETTER



school of a

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	1	2	3	4	chars	order
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	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

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

df2

- To modify columns labels use: **column=**
- if indexes or columns were strings we could use for example:
index= str.lower()

	0	1	2	3	4	chars	order
zero	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



school of a

3- Data Transformation

Discretization

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

ser2

0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8

dtype: int64

0 doesn't
Belong
to Any
category

0	NaN
1	(0, 4]
2	(0, 4]
3	(0, 4]
4	(0, 4]
5	(4, 7]
6	(4, 7]
7	(4, 7]
8	(7, 9]

dtype: category

Categories (3, interval[int64]): [(0, 4] < (4, 7] < (7, 9]]

The values are grouped in 3
categories: $0 \rightarrow 4$, $5 \rightarrow 7$, $8 \rightarrow 9$
==

$(0,4],(4,7],(7,9]$

- "(" means the value is **out**. The
"]" means the value is **in**.

```
1 # grouping the ser2 values into four categories with the same length  
2 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

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

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



school of ai

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

	0	1	2	3	4	chars	order
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

array([0, 2, 1])

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)

```
0    0
6    6
7    7
3    3
1    1
dtype: int64
```



school of a

4- String manipulation

String methods

- String objects have useful methods that can be used:

```
1 # split a string specifying a separator
2 myStr= "This is an example"
3 splitted=myStr.split(" ")
```

['This', 'is', 'an', 'example']

```
1 # join strings with a separator
2 "-".join(splitted)
```

'This-is-an-example'

```
1 # replace a value in string by another value
2 newStr=myStr.replace(" ", "_")
```

'This_is_an_example'

```
1 # find a value in a string using find
2 myStr.find("e")
```

11

If "e" doesn't exist it
will return -1

```
1 # find a value in a string using index
2 myStr.index("e")
```

If "e" doesn't exist it
will raise an exception

```
1 # find a value in a string using in
2 "e" in myStr
```

3

```
1 # the number of substring in a string
2 myStr.count(" ")
```



school of a

References

- SQLAlchemy authors and contributors. Sqlalchemy 1.2 documentation. On-line at <https://docs.sqlalchemy.org/en/latest/core/dml.html>. Accessed 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] pysheet. Python sqlalchemy cheatsheet. On-line at <https://pysheet.readthedocs.io/en/latest/notes/python-sqlalchemy.html>. Accessed on 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