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

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**

On disk Files

- You have just to **choose** the right **function** to use with the right **arguments**:

```
#viewing the content of A3P-w2-ex1.csv
!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 **‘,’**

No need to specify
a header or a separator

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

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

```
f2= pd.read_csv("A3P-w2-ex1.csv", header=None, sep=",")
f2
```

In this case,no need to specify a separator

	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

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]

1- Data Files: Reading and Writing

On disk Files

```
!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**

```python
header=[ "col"+str(i) for i in range(1,6)]
ind= header[len(header)-1]
f3= pd.read_csv("A3P-w2-ex2.csv", names=header,index_col=ind)
f3
```

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

	col1	col2	col3	col4
col5				
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

[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
one,1,2,3,4,NA
two,5,6,,8,world
three,9,10,11,12,foo
```

A missing value

```
pd.read_csv("A3P-w2-ex5.csv")
```

	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

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

```
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

[By Amina Delali]

1- Data Files: Reading and Writing

On disk Files

```python
# 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)

for i, chunk in zip(range(1,6),tfr):
  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
```

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

```
(10000, 5)
```

Chunksize== **2** rows

We read only **10** rows (from 10000)

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

[By Amina Delali]

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}]
```

```
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	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

[By Amina Delali]

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

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

1- Data Files: Reading and Writing

On disk Files

Will read all the tables

```python
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

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

[By Amina Delali]

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

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

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

Saving the files to different files format

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

Content of file1.txt and file1.csv

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

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:

```
from google.colab import files
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

[By Amina Delali]

1- Data Files: Reading and Writing

Web Interaction

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

```
import requests
# url to get the first page of 30 issues of a GitHub Repository
url1 = "https://api.github.com/repos/pandas-dev/pandas/issues"
# 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"


alliss= requests.get(url1)
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

```
alliss= requests.get(url1)
closed= requests.get(url2)

# create DataFrames from the responses of the request
da= alliss.json()
daf=pd.DataFrame(da,columns=["id","state"])
# create DataFrames from
dc= closed.json()
dcf=pd.DataFrame(dc,columns=["id","state"])

print(daf.shape)
print(dcf.shape)
```

```
(100, 2)
```

```
(30, 2)
```

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

The two columns we selected

DataBase Interaction

- 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")

# to use "meta" later for the creation of the Table
meta = MetaData(DB)

# define a table's scheme
myTable=Table("myTab",meta,Column("id",types.String),Column("value",types.Integer))
# creation of the table
myTable.create()
# verification if the table exist
DB.has_table("myTab")
```

```
True
```

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

[By Amina Delali]

DataBase Interaction

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

insertion= myTable.insert().values([{"id": "a", "value":5},{"id": "b", "value":10},{"id": "c",
# insert the values nto the table
DB.execute(insertion)
```

Value to insert with the corresponding column

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
```

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

	id	value
0	a	5
1	b	10
2	c	10

[By Amina Delali]

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

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

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

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

```python
# drop the na values
ser1.dropna()
```

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

The new Series will contain only those two values

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

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

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

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

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

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

[By Amina Delali]

2- Missing data

Filling in

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

```
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**

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

axis=0

	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

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

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

df1 will be modified

	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

[By Amina Delali]

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

```
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

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

	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

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

Transforming data

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

	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

	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

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

```
# create a mapping using a dict
myMap ={ "a":"First letter","c":"Third letter","b":"Second letter"}
# create a new column using that mapping
df1["order"]=df1["chars"].map(myMap)
```

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

order
FIRST LETTER
SECOND LETTER
THIRD LETTER

3- Data Transformation

[By Amina Delali]

3- Data Transformation

Replacing values and Renaming indexes

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

df2

	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

- to modify only one value:
df1.replace(0.99,1)
- using **inplace=True**, will **modify** the **original** DataFrame

```
# replacing 0.99 values by 1 and  12 by 120 using a dictionary
df2=df1.replace({0.99:1,12:120})
```

3 doesn't exist

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

df2

	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

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

[By Amina Delali]

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 → 4 ,5 → 7, 8 → 9
==
(0,4],(4,7],(7,9]
- "**(**" means the value is **out**. The "**]**" means the value is **in**.

```
# 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
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.

3- Data Transformation

Permutation

```
# create an array with not ordred range values  (permuted)
norder= np.random.permutation(3)
```

```
array([0, 2, 1])
```

```
# creating a new df reordred
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

Random sampling

```
# selecting randomly 5 values from ser2
ser2.sample(n=5)
```

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

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

4- String manipulation

String methods

- String object have useful methods that can be used:

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

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

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

```
'This-is-an-example'
```

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

```
'This_is_an_example'
```

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

```
11
```

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

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

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

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

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

```
3
```

[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