
Programming and frameworks for ML

Python for data analysis

About Me

Big Data Consultant at Santander / Big Data Lecturer

- More than 20 years of experience in different environments, technologies, customers, countries ...
- Passionate about data and technology
- Enthusiastic about Big Data world and NoSQL



Daniel Villanueva Jiménez

Arquitecto de Datos at Santander Tecnología

Greater Madrid Metropolitan Area · **500+ connections** ·



Santander Tecnología



Universidad Pontificia de
Salamanca



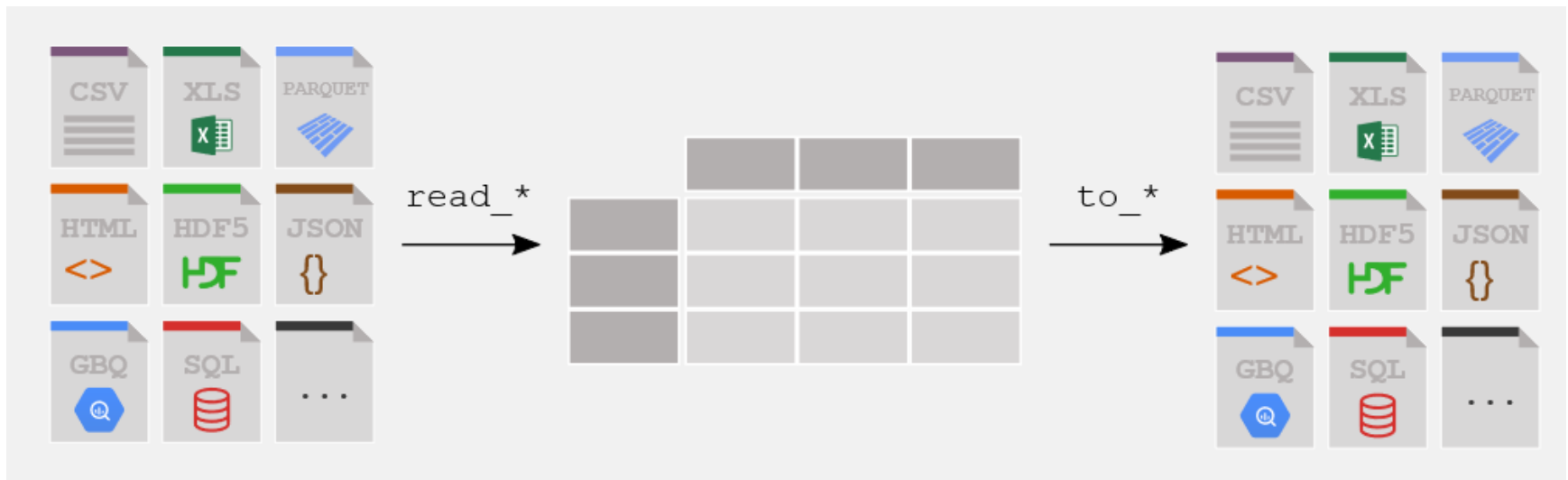
Agenda

- Reading / Writing data
- Exploring a DataFrame
- Renaming headings
- Filtering Columns
- Filtering Rows
- Sorting Data
- Changing Data
- Deleting Data
- Grouping Data
- Concatenating Data
- Joining Data
- Pivot Tables



Reading / Writing data

- Pandas supports the integration with many file formats or data sources out of the box (csv, excel, sql, json, parquet,...).



Reading data in text format

- The function **pd.read_csv()** allows you to read a file and store it in a DataFrame
- With the default options, files must have a header (first row) and the separator is a comma
- The file could be both on disk and on the network
- The file can be compressed!

```
pd.read_csv("Sacramento-RealEstate-Transactions.csv")  
pd.read_csv("https://raw.githubusercontent.com/Sacramento-RealEstate-Transactions.csv")  
pd.read_csv("Sacramento-RealEstate-Transactions.gz")
```

Reading data in text format

- Sometimes the separator is something different to a comma, for example a pipe (|).
- Use the **sep** argument to change the separator

```
%%writefile input_data.txt  
a|b|c|d|message  
1|2|3|4|hello  
5|6|7|8|world  
9|10|11|12|foo
```

Writing input_data.txt

```
pd.read_csv("input_data.txt", sep = "|")
```

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

Reading data in text format

- We can use the **header** parameter to tell pandas where the header is located (None if the data don't have a header)

```
%%writefile input_data.txt  
1|2|3|4|hello  
5|6|7|8|world  
9|10|11|12|foo
```

Overwriting input_data.txt

```
pd.read_csv("input_data.txt", sep = "|", header = None)
```

	0	1	2	3	4
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

Reading data in text format

- If the file don't have a header we can specify the columns' names with the **names** parameter

```
%%writefile input_data.txt  
1|2|3|4|hello  
5|6|7|8|world  
9|10|11|12|foo
```

Overwriting input_data.txt

```
pd.read_csv("input_data.txt", sep = "|", header = None,  
            names = ["a", "b", "c", "d", "names"])
```

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

Reading data in text format

- The **na_values** parameter specifies the null values forming

```
%%writefile input_data.txt  
a|b|c|d|message  
1|2|3|NA|hello  
5|6|7|8|world  
9|NA|11|12|foo
```

Overwriting input_data.txt

```
pd.read_table("input_data.txt", sep = '|', na_values=['NA'])
```

	a	b	c	d	message
0	1	2.0	3	NaN	hello
1	5	6.0	7	8.0	world
2	9	NaN	11	12.0	foo

Reading data in text format

- The **pd.read_fwf()** function allows you to read a file when the columns have fixed positions

```
%%writefile input_data.txt  
a b c d message  
1 2 223 4 hello  
5 6 7 8 world  
9 10 11 12 foo
```

Overwriting input_data.txt

```
pd.read_fwf("input_data.txt")
```

	a	b	c	d	message
0	1	2	223	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

Reading data in text format

- The **colspecs** parameter allows you specify the actual positions for the columns

```
%writefile input_data.txt  
a b c d message  
1 2 3 4 hello  
5 6 5 8 world  
9 10 11 12 foo
```

Overwriting input_data.txt

```
colspecs = [(0, 2), (3, 5), (6, 8), (13, 30)]  
pd.read_fwf("input_data.txt", colspecs=colspecs)
```

	a	b	c	message
0	1	2	3	hello
1	5	6	5	world
2	9	10	11	foo

Reading data in text format

- The **converters** parameter allows you to set conversion functions in the columns of the DataFrame

```
%writefile input_data.txt  
col1|col2|col3  
one|1.232,53|a  
two|2.000,00|b
```

Writing input_data.txt

```
pd.read_csv("input_data.txt", sep = "|",  
            converters = {'col2': lambda value: float(value.replace('.', '').replace(',', '.'))}  
            )
```

	col1	col2	col3
0	one	1232.53	a
1	two	2000.00	b

Reading data from Excel

- Pandas also allows you to read an Excel format file
- If we want to read several sheets of the same Excel file, it is convenient to first load the file in memory with the **pd.ExcelFile()** method

```
dataframe = pd.read_excel("FL_insurance_sample.xlsx")  
dataframe = pd.read_excel("FL_insurance_sample.xlsx", 'FL_insurance_sample')
```

```
xlsx = pd.ExcelFile("FL_insurance_sample.xlsx")  
dataframe = pd.read_excel(xlsx, 'FL_insurance_sample')
```

Reading data from a JSON file

- Using the **pd.read_json()** function, pandas will **read** data in JSON format and load it into a DataFrame

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

Overwriting input_data.json

```
pd.read_json("input_data.json")
```

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

Reading data from a JSON file

- An alternative is to use the **json** library with the Pandas **Dataframe** function

```
%%writefile input_data.json
{
  "name": "Wes",
  "places_lived": ["United States", "Spain", "Germany"],
  "siblings": [
    {"age": 30, "name": "Scott", "pets": ["Zeus", "Zuko"]},
    {"age": 38, "name": "Katie", "pets": ["Sixes", "Stache", "Cisco"]}
  ]
}
```

Overwriting input_data.json

```
import json

with open('input_data.json') as json_data:
    result = json.load(json_data)

pd.DataFrame(result['siblings'], columns=['name', 'age'])
```

	name	age
0	Scott	30
1	Katie	38

Reading data from a Web Service

- To get data from a web service we could use the **request** library

```
import requests
```

```
url = 'https://api.github.com/repos/pandas-dev/pandas/issues'  
resp = requests.get(url)
```

```
if resp.ok:  
    data = resp.json()  
    dataframe = pd.DataFrame(data, columns=['number', 'title', 'labels', 'state'])  
  
dataframe.head()
```

	number	title	labels	state
0	21649	DOC: Fix versionadded directive typos in Inter...	[{'id': 134699, 'node_id': 'MDU6TGFiZWwxMzQ2OT...}	open
1	21648	API: Categorical.unique() should not drop unus...	[]	open
2	21647	addresses GH #21646	[]	open
3	21646	Test fixture datapath uses relative instead of...	[]	open
4	21645	ENH: add return_inverse to df.duplicated	[{'id': 76812, 'node_id': 'MDU6TGFiZWw3NjgxMg=...}	open



Reading data from HTML

- Pandas allows to read a file with HTML format through the **read_html()** function

Politics [\[edit \]](#)

Main article: *Politics of Minnesota*

See also: *List of political parties in Minnesota, United States congressional delegations from Minnesota, Minnesota's congressional districts, and Political party strength in Minnesota*

Minnesota is known for a politically active citizenry, and [populism](#) has been a long-standing force among the state's [political parties](#).^{[150][151]} Minnesota has a consistently high [voter turnout](#). In the [2008 U.S. presidential election](#), 78.2% of eligible Minnesotans voted – the highest percentage of any U.S. state – versus the national average of 61.2%.^[152] Voters can register on [election day](#) at their [polling places](#) with evidence of residency.^[153]

[Hubert Humphrey](#) brought national attention to the state with his address at the [1948 Democratic National Convention](#). Minnesotans have consistently cast their Electoral College votes for Democratic presidential candidates since 1976, longer than any other state. Minnesota is the only state in the nation that did not vote for [Ronald Reagan](#) in either of his presidential runs. Minnesota has gone for the Democratic Party in every presidential election since 1960, with the exception of 1972, when it was carried by Republican [Richard Nixon](#).

Both the Democratic and Republican parties have major-party status in Minnesota, but its state-level Democratic party has a different name, officially known as the [Minnesota Democratic-Farmer-Labor Party](#) (DFL). It was formed out of a 1944 alliance of the Minnesota Democratic and [Farmer-Labor](#) parties.

The state has had active third-party movements. The [Reform Party](#), now the [Independence Party](#), was able to elect former mayor of [Brooklyn Park](#) and [professional wrestler Jesse Ventura](#) to the governorship in 1998. The

Election results from statewide races^[149]

Year	Office	GOP	DFL	Others
2020	President	45.3%	52.4%	2.3%
	Senator	43.5%	48.8%	7.7%
2018	Governor	42.4%	53.9%	3.7%
	Senator	36.2%	60.3%	3.4%
	Senator	42.4%	53.0%	4.6%
2016	President	44.9%	46.4%	8.6%
2014	Governor	44.5%	50.1%	5.4%
	Senator	42.9%	53.2%	3.9%
2012	President	45.1%	52.8%	2.1%
	Senator	30.6%	65.3%	4.1%
2010	Governor	43.2%	43.7%	13.1%



Reading data from HTML

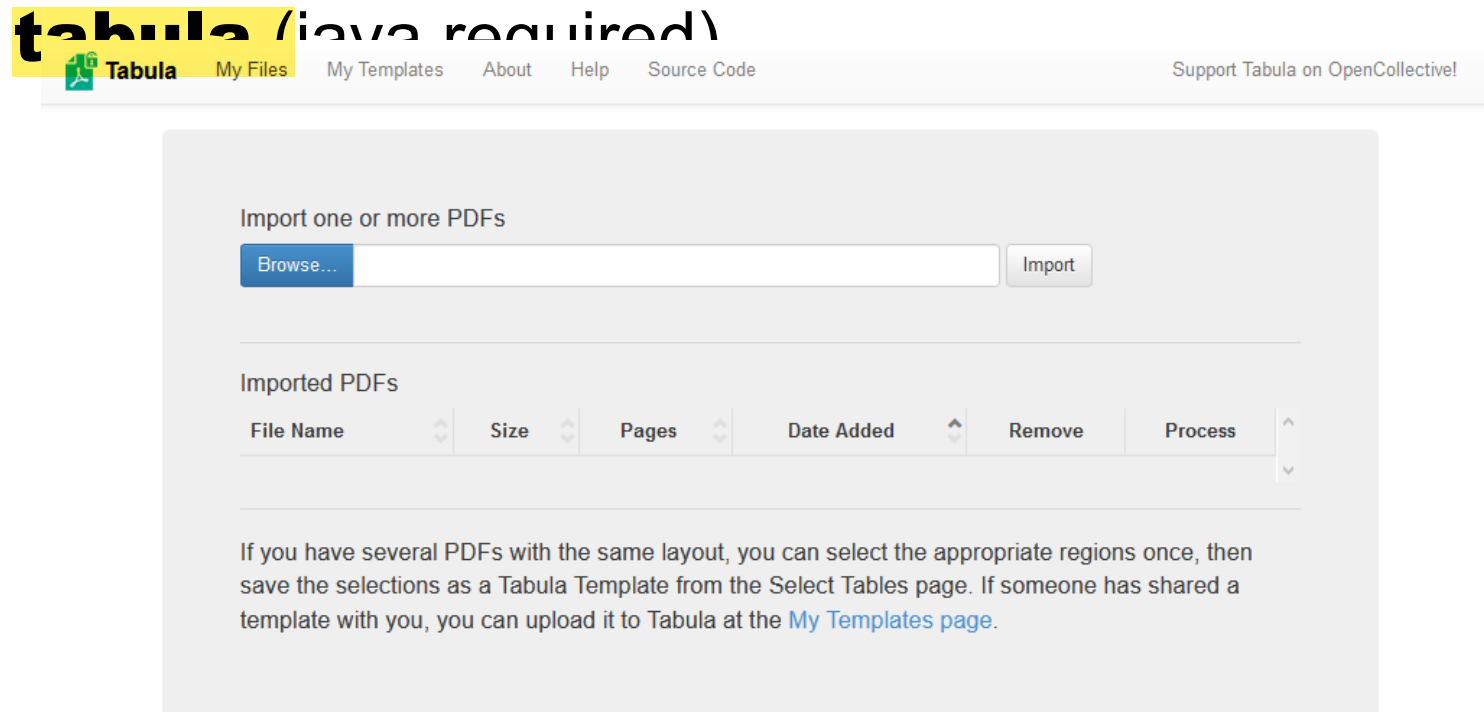
- This function returns a list of dataframes (there may be several tables on the website)

```
dataframes = pd.read_html('https://en.wikipedia.org/wiki/Minnesota',  
                           match='Election results from statewide races')  
  
dataframes[0].head(5)
```

	Year	Office	GOP	DFL	Others
0	2020	President	45.3%	52.4%	2.3%
1	2020	Senator	43.5%	48.8%	7.7%
2	2018	Governor	42.4%	53.9%	3.7%
3	2018	Senator	36.2%	60.3%	3.4%
4	2018	Senator	42.4%	53.0%	4.6%

Reading data from PDF

- There are several options to read data from PDF format
- One option is to use a open source utility called **tabula** (java required)



Reading data from PDF

[My Files](#)
[My Templates](#)
[About](#)
[Help](#)
[Source Code](#)

[Support Tabula on OpenCollective!](#)

1

Sacramento-RealEstate-Transactions.h...

Templates

Clear All Selections

Autodetect Tables

Preview & Export Extracted Data

1.

4/23/2021

Sacramento-RealEstate-Transactions.html

	street	city	zip	state	beds	baths	sqft	type	sale_date	price	latitude	longitude
0	3526 HIGH ST	SACRAMENTO	95838	CA	2	1	836	Residential	Wed May 21 00:00:00 EDT 2008	59222	38.631913	-121.434879
1	51 OMAHA CT	SACRAMENTO	95823	CA	3	1	1167	Residential	Wed May 21 00:00:00 EDT 2008	68212	38.478902	-121.431028
2	2796 BRANCH ST	SACRAMENTO	95815	CA	2	1	796	Residential	Wed May 21 00:00:00 EDT 2008	68880	38.618305	-121.443839
3	2803 JANETTE WAY	SACRAMENTO	95815	CA	2	1	852	Residential	Wed May 21 00:00:00 EDT 2008	69307	38.616835	-121.439144
4	6001 MCMAHON DR	SACRAMENTO	95824	CA	2	1	797	Residential	Wed May 21 00:00:00 EDT 2008	81900	38.519470	-121.435768
5	5828 PEPPERMILL CT	SACRAMENTO	95841	CA	3	1	1122	Condo	Wed May 21 00:00:00 EDT 2008	89921	38.662595	-121.327818
6	6048 OGDEN NASH WAY	SACRAMENTO	95842	CA	3	2	1104	Residential	Wed May 21 00:00:00 EDT 2008	80895	38.681659	-121.351709
7	2561 19TH AVE	SACRAMENTO	95820	CA	3	1	1177	Residential	Wed May 21 00:00:00 EDT 2008	91002	38.535092	-121.481367
8	11130 TRINITY RIVER DR Unit 114	RANCHO CORDOVA	95670	CA	2	2	941	Condo	Wed May 21 00:00:00 EDT 2008	94905	38.621188	-121.270555
9	7325 10TH ST	RIO LINDA	95673	CA	3	2	1146	Residential	Wed May 21 00:00:00 EDT 2008	98917	38.700909	-121.442979
10	645 MORRISON AVE	SACRAMENTO	95838	CA	3	2	909	Residential	Wed May 21 00:00:00 EDT 2008	100309	38.637663	-121.451523
11	4085 FAWN CIR	SACRAMENTO	95823	CA	3	2	1289	Residential	Wed May 21 00:00:00 EDT 2008	106250	38.470746	-121.458918
12	2930 LA ROSA RD	SACRAMENTO	95815	CA	1	1	871	Residential	Wed May 21 00:00:00 EDT 2008	106852	38.618698	-121.435833
13	2113 KIRK WAY	SACRAMENTO	95822	CA	3	1	1020	Residential	Wed May 21 00:00:00 EDT 2008	107502	38.482215	-121.492608
14	4533 LOCH HAVEN WAY	SACRAMENTO	95842	CA	2	2	1022	Residential	Wed May 21 00:00:00 EDT 2008	108750	38.672914	-121.359348
15	7340 HAMDEN PL	SACRAMENTO	95842	CA	2	2	1134	Condo	Wed May 21 00:00:00 EDT 2008	107900	38.700651	-121.351279
16	6715 6TH ST	RIO LINDA	95673	CA	2	1	844	Residential	Wed May 21 00:00:00 EDT 2008	113263	38.689591	-121.452239
17	6236 LONGFORD DR Unit 1	CITRUS HEIGHTS	95621	CA	2	1	795	Condo	Wed May 21 00:00:00 EDT 2008	116250	38.679776	-121.314089
18	250 PERALTA AVE	SACRAMENTO	95833	CA	2	1	588	Residential	Wed May 21 00:00:00 EDT 2008	120000	38.612099	-121.469099
19	113 LEEWILL AVE	RIO LINDA	95673	CA	3	2	1356	Residential	Wed May 21 00:00:00 EDT 2008	121630	38.689999	-121.463229

file:///C:/Develop/Projects/Practice-Pandas/notebooks/Sacramento-RealEstate-Transactions.html

1/1

Reading data from PDF

Tabula

My Files

My Templates

About

Help

Source Code

Support Tabula on OpenCollective!

Is the extracted data incorrect?

You can revise your selected cells or try an alternate extraction method.

Revise Selected Cells

Data has been extracted from the cells you selected in the previous step. You can revise your selection(s) to add or remove cells.

← Revise selection(s)

Choose Alternate Extraction Method

The current preview uses the **Stream** extraction method. If the data is not mapped to the correct cells, try the **Lattice** method instead.

Stream

Lattice

Stream looks for *whitespace* between columns, while **Lattice** looks for *boundary lines* between columns.

Still look wrong?

Contact the developers and tell us what you tried to do that didn't work.

Sacramento-RealEstate-Transactions.h...

Export Format: CSV

Export

Copy to Clipboard

Preview of Extracted Tabular Data

	street	city	zip	state	beds	baths	sqft	type	sale_date	price	latitude	longitude
0	3526 HIGH ST	SACRAMENTO	95838	CA	2	1	836	Residential	Wed May 21 00:00:00 EDT 2008	59222	38.631913	-121.434879
1	51 OMAHA CT	SACRAMENTO	95823	CA	3	1	1167	Residential	Wed May 21 00:00:00 EDT 2008	68212	38.478902	-121.431028
2	2796 BRANCH ST	SACRAMENTO	95815	CA	2	1	796	Residential	Wed May 21 00:00:00 EDT 2008	68880	38.618305	-121.443839
3	2805 JANETTE WAY	SACRAMENTO	95815	CA	2	1	852	Residential	Wed May 21 00:00:00 EDT 2008	69307	38.616835	-121.439146
4	6001 MCMAHON DR	SACRAMENTO	95824	CA	2	1	797	Residential	Wed May 21 00:00:00 EDT 2008	81900	38.519470	-121.435768
5	5828 PEPPERMILL CT	SACRAMENTO	95841	CA	3	1	1122	Condo	Wed May 21 00:00:00 EDT 2008	89921	38.662595	-121.327813
6	6048 OGDEN NASH WAY	SACRAMENTO	95842	CA	3	2	1104	Residential	Wed May 21 00:00:00 EDT 2008	90895	38.681659	-121.351705
7	2561 19TH AVE	SACRAMENTO	95820	CA	3	1	1177	Residential	Wed May 21 00:00:00 EDT 2008	91002	38.535092	-121.481367
8	11150 TRINITY RIVER DR Unit 114	RANCHO CORDOVA	95670	CA	2	2	941	Condo	Wed May 21 00:00:00 EDT 2008	94905	38.621188	-121.270555
9	7325 10TH ST	RIO LINDA	95673	CA	3	2	1146	Residential	Wed May 21 00:00:00 EDT 2008	98937	38.700909	-121.442979
10	645 MORRISON AVE	SACRAMENTO	95838	CA	3	2	909	Residential	Wed May 21 00:00:00 EDT 2008	100309	38.637663	-121.451520
11	4085 FAWN CIR	SACRAMENTO	95823	CA	3	2	1289	Residential	Wed May 21 00:00:00 EDT 2008	106250	38.470746	-121.458918
12	2930 LA ROSA RD	SACRAMENTO	95815	CA	1	1	871	Residential	Wed May 21 00:00:00 EDT 2008	106852	38.618698	-121.435833
13	2113 KIRK WAY	SACRAMENTO	95822	CA	3	1	1020	Residential	Wed May 21 00:00:00 EDT 2008	107502	38.482215	-121.492603
14	4533 LOCH HAVEN WAY	SACRAMENTO	95842	CA	2	2	1022	Residential	Wed May 21 00:00:00 EDT 2008	108750	38.672914	-121.359340
15	7340 HAMDEN PL	SACRAMENTO	95842	CA	2	2	1134	Condo	Wed May 21 00:00:00 EDT 2008	110700	38.700051	-121.351278
16	6715 6TH ST	RIO LINDA	95673	CA	2	1	844	Residential	Wed May 21 00:00:00 EDT 2008	113263	38.689591	-121.452239
17	6236 LONGFORD DR Unit 1	CITRUS HEIGHTS	95621	CA	2	1	795	Condo	Wed May 21 00:00:00 EDT 2008	116250	38.679776	-121.314089
18	250 PERALTA AVE	SACRAMENTO	95833	CA	2	1	588	Residential	Wed May 21 00:00:00 EDT 2008	120000	38.612099	-121.469095
19	113 LEEWILL AVE	RIO LINDA	95673	CA	3	2	1356	Residential	Wed May 21 00:00:00 EDT 2008	121630	38.689999	-121.463220

21

Reading data



🔍 pandas read |



🔍 pandas read **google sheet**

🔍 pandas read **excel sheet**

🔍 pandas read **csv**

🔍 pandas read **json**

🔍 pandas read **text**

🔍 pandas read **sql**

🔍 pandas read **file**

🔍 pandas read **parquet**

🔍 pandas read **text file**

🔍 pandas read **csv example**

Denunciar predicciones inadecuadas
[Más información](#)

tas



Writing Data

- Once we have a DataFrame in memory, we could write it to disk with one of the following functions:
 - dataframe. **to_csv**("file.csv")
 - dataframe. **to_excel**("file.xlsx")
 - dataframe. **to_json**("file.json")

```
dataframe = pd.read_excel("FL_insurance_sample.xlsx")
dataframe = pd.read_excel("FL_insurance_sample.xlsx", 'FL_insurance_sample')
dataframe = pd.read_json("FL_insurance_sample.json")
```

```
xlsx = pd.ExcelWriter("file.xlsx")
dataframe1.to_excel(xlsx, 'Sheet1')
dataframe1.to_excel(xlsx, 'Sheet1', index = False)
xlsx.save()
```

Working with databases

- The **sqlalchemy** package allows you to connect to a relational database
- We have to create an **engine** to connect with the database and use the appropriate connecting string

```
import pandas as pd
from sqlalchemy import create_engine
```

```
engine = create_engine('mysql://scott:tiger@localhost/test')
engine = create_engine('postgresql://scott:tiger@localhost:5432/mydatabase')
engine = create_engine('sqlite:///memory:', echo=False)
```



Working with databases

- **pd.to_sql()** function allows you to save a Pandas Dataframe to the database

```
%%writefile input_data.txt
a|b|c|d|message
1|2|3|4|hello
5|6|7|8|world
9|10|11|12|foo
```

Overwriting input_data.txt

```
df = pd.read_csv("input_data.txt", sep = "|")
df
```

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

```
engine = create_engine('sqlite:///database.db', echo=False)
```

```
df.to_sql('table', engine, if_exists = 'append', index = False)
```

Working with databases

- **`pd.read_sql_table()`** and **`pd.read_sql_query()`** allow you to read data from a database to a Pandas Dataframe

```
pd.read_sql_table("table", engine)
```

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

```
pd.read_sql_query("select * from 'table' where a <= 5", engine)
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world

Working with databases

- **Jupyter** and the **ipython_sql** extension lets you work with SQL and Pandas

```
%load_ext sql
%sql sqlite:///database.db
```

```
%%sql
```

```
select * from 'table' where a <= 5
```

```
* sqlite:///database.db
```

```
Done.
```

```
a  b  c  d  message
```

```
1  2  3  4    hello
```

```
5  6  7  8    world
```

```
df=%sql select * from 'table' where a <= 5
```

```
* sqlite:///database.db
```

```
Done.
```

```
df
```

```
a  b  c  d  message
```

```
1  2  3  4    hello
```

```
5  6  7  8    world
```

Pickle data format

- **Picket** is a binary **file format** for storing python objects.
- Faster than CSV (~200%) and smaller (~50%)
- Keeps the information about data types

```
df.to_pickle("data.pickle")
```

```
pd.read_pickle("data.pickle")
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world

```
df
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world

Feather data format

- **Feather** is a portable **file format** for storing data frames (from languages like Python or R)

```
import feather
```

```
feather.write_dataframe(df, 'test.feather')  
feather.read_dataframe("test.feather")
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world

```
feather.read_dataframe("cars.feather")
```

	speed	dist
0	4.0	2.0
1	4.0	10.0
2	7.0	4.0
3	7.0	22.0
4	9.0	16.0

df

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world



Feather data format

- We can save data in Python and reading the same data in R

```
library(arrow)
```

```
df <- read_feather("test.feather")  
df
```

A tibble: 2 × 5

a	b	c	d	message
<int>	<int>	<int>	<int>	<chr>
1	2	3	4	hello
5	6	7	8	world

```
write_feather(cars, "cars.feather")
```

```
head(cars)
```

A data.frame: 6 × 2

	speed	dist
	<dbl>	<dbl>
1	4	2
2	4	10
3	7	4
4	7	22
5	8	16
6	9	10

Fake Data

- There are times when we need to quickly generate fake data

```
from faker import Faker

def create_rows(num=1):
    fake = Faker()
    output = [{"name":fake.name(),
               "address":fake.address(),
               "name":fake.name(),
               "email":fake.free_email(),
               "address":fake.address(),
               "city":fake.city(),
               "state":fake.state()} for x in range(num)]
    return output

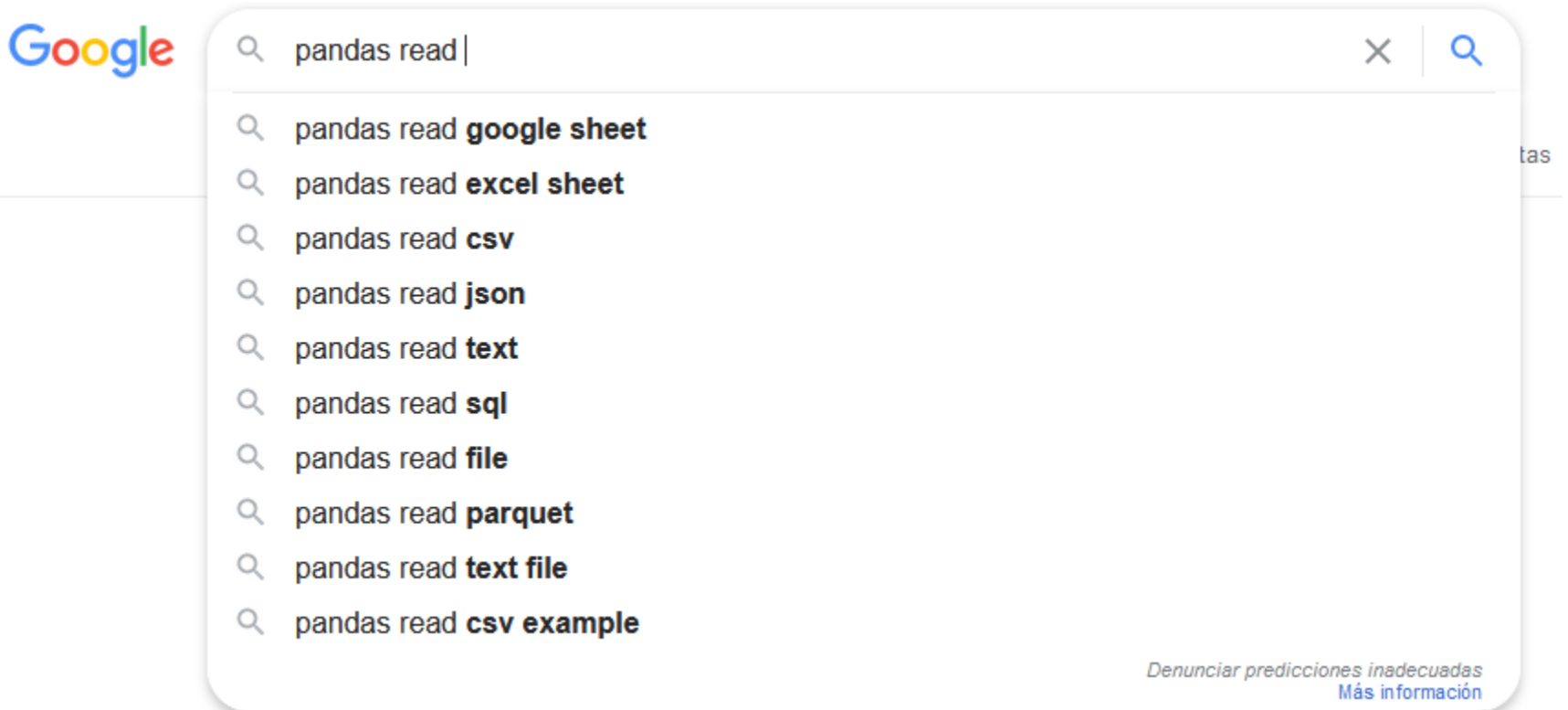
df = pd.DataFrame(create_rows(10))
```

df					
	name	address	email	city	state
0	Jeffrey Aguirre	PSC 6238, Box 6681\nAPO AA 79034	lisa48@gmail.com	Lake Melanie	Mississippi
1	Brenda Lewis	678 Frederick Mews\nMillershire, AR 03554	johnwells@yahoo.com	Moralesview	Nevada
2	Matthew Williams	696 Elizabeth Mission Suite 478\nLake Alicevie...	thernandez@gmail.com	North Jimmybury	Maryland
3	Michelle Powers	014 Snyder Squares Suite 211\nWest Patricia, V...	areese@yahoo.com	Foleychester	Tennessee
4	Dawn Miller	815 David Circles\nVargasberg, WY 39498	lindaday@gmail.com	East Johnbury	Tennessee
5	Katie Hale	87663 Elizabeth Circles\nGarciahire, UT 81092	garrettwilliams@hotmail.com	South Amandaberg	Louisiana
6	Thomas Clark	6942 Gallagher Squares\nHannahshire, MT 36652	mariamartinez@yahoo.com	New Sonyaton	Utah
7	Kelly Fisher	55835 Scott Circles\nKimberlyview, CT 52635	erogers@yahoo.com	North Timothyport	South Carolina
8	Sharon Castro	68617 Eric Ridge Suite 348\nNew Danahaven, PA ...	jeffreymartinez@yahoo.com	Cassandraview	West Virginia
9	Victoria Pitts	48143 Jose Cliffs\nEast Andrew, TN 86824	nrussell@yahoo.com	Holttown	Vermont



Other sources

- We can read data from practically any source



Exercise 1

- Load the information from the following url into a Dataframe called 'df1':

<https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv>

- Load in a Dataframe called 'df2', the information of the following url:

<https://raw.githubusercontent.com/justmarkham/DAT8/master/data/u.user>

- The column 'user_id' must be the index of the DataFrame

order_id	quantity	item_name	choice_description	item_price	
0	1	1	Chips and Fresh Tomato Salsa	NaN	\$2.39
1	1	1	Izze	[Clementine]	\$3.39
2	1	1	Nantucket Nectar	[Apple]	\$3.39
3	1	1	Chips and Tomatillo-Green Chili Salsa	NaN	\$2.39
4	2	2	Chicken Bowl	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	\$16.98

	age	gender	occupation	zip_code
user_id				
1	24	M	technician	85711
2	53	F	other	94043
3	23	M	writer	32067
4	24	M	technician	43537
5	33	F	other	15213

Exercise 2

- Write the dataframes from the previous exercise in an excel called "Data.xlsx"
 - 'df1' save it on a sheet called 'chipotle' (without the index)
 - 'df2' on another sheet called 'user'
- Recover in a different DataFrame the information from the 'user' sheet of the excel file "Data.xlsx".

	age	gender	occupation	zip_code
user_id				
1	24	M	technician	85711
2	53	F	other	94043
3	23	M	writer	32067
4	24	M	technician	43537
5	33	F	other	15213

Exercise 3

- Read the 'Provincias' data from <https://geoportalgasolineras.es/#/Descargas>

Inicio Envío precios Histórico precios **Descargar ficheros** Planes de descuento Informes mensuales Premios Geoportales europeos

Descargar ficheros

Por favor, seleccione el tipo de fichero que desea descargar:

 Dispositivos GPS
  Estaciones de servicio y postes marítimos en KML
  Información sobre planes de descuento
  Información actualizada de precios

Ficheros de información actualizada de precios



Tipo de búsqueda *: Estaciones de servicio
 Ubicación *: Toda España
 Tipo de carburante *: Gasolina 95 E5
 Descargar

[Descarga de precios de todos los carburantes \(estaciones de servicio\)](#)
[Descarga de precios de todos los carburantes \(postes marítimos\)](#)

Servicios REST

Servicio	URL	Ayuda
Estaciones de servicio	https://sedeaplicaciones.minetur.gob.es/ServiciosRESTCarburantes/PreciosCarburantes/EstacionesTerrestres/	
Estaciones de servicio por comunidad	https://sedeaplicaciones.minetur.gob.es/ServiciosRESTCarburantes/PreciosCarburantes/EstacionesTerrestres/FichaCCAA/IDCAA/	

Agenda

- Reading / Writing data
- **Exploring a DataFrame**
- Renaming headings
- Filtering Columns
- Filtering Rows
- Sorting Data
- Changing Data
- Deleting Data
- Grouping Data
- Concatenating Data
- Joining Data
- Pivot Tables



Exploring a DataFrame

- **df.info()** shows summary information about the DataFrame

```
dataframe.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype  
---  -
 0   model   32 non-null     object 
 1   mpg     32 non-null     float64
 2   cyl     32 non-null     int64  
 3   disp    32 non-null     float64
 4   hp      32 non-null     int64  
 5   drat    32 non-null     float64
 6   wt      32 non-null     float64
 7   qsec    32 non-null     float64
 8   vs      32 non-null     int64  
 9   am      32 non-null     int64  
10   gear    32 non-null     int64  
11   carb    32 non-null     int64  
dtypes: float64(5), int64(6), object(1)
memory usage: 3.1+ KB
```

Exploring a DataFrame

- **df.count()** returns a Serie with the number of **non-null** values for each column

```
dataframe.count()
```

```
model      32  
mpg        32  
cyl        32  
disp       32  
hp         32  
drat       32  
wt         32  
qsec       32  
vs         32  
am         32  
gear       32  
carb       32  
dtype: int64
```

```
dataframe.count(axis = 1)
```

```
0      12  
1      12  
2      12  
3      12  
4      12  
5      12  
6      12  
7      12  
8      12  
9      12  
10     12  
11     12  
12     12  
13     12  
14     12  
15     12  
16     12
```

		axis columns		
		0	1	2
axis index	0	0,0	0,1	0,2
	1	1,0	1,1	1,2
	2	2,0	2,1	2,2

Exploring a DataFrame

- **df.describe()** returns a DataFrame with statistical information of each of the numerical columns

```
dataframe.describe()
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
count	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.0000
mean	20.090625	6.187500	230.721875	146.687500	3.596563	3.217250	17.848750	0.437500	0.406250	3.687500	2.8125
std	6.026948	1.785922	123.938694	68.562868	0.534679	0.978457	1.786943	0.504016	0.498991	0.737804	1.6152
min	10.400000	4.000000	71.100000	52.000000	2.760000	1.513000	14.500000	0.000000	0.000000	3.000000	1.0000
25%	15.425000	4.000000	120.825000	96.500000	3.080000	2.581250	16.892500	0.000000	0.000000	3.000000	2.0000
50%	19.200000	6.000000	196.300000	123.000000	3.695000	3.325000	17.710000	0.000000	0.000000	4.000000	2.0000
75%	22.800000	8.000000	326.000000	180.000000	3.920000	3.610000	18.900000	1.000000	1.000000	4.000000	4.0000
max	33.900000	8.000000	472.000000	335.000000	4.930000	5.424000	22.900000	1.000000	1.000000	5.000000	8.0000

Exploring a DataFrame

- **df.describe()** only describe numerical columns but can be fixed with the “**include**” parameter

```
dataframe.describe(include='all')
```

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
count	32	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.0000
unique	32	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	Honda Civic	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	20.090625	6.187500	230.721875	146.687500	3.596563	3.217250	17.848750	0.437500	0.406250	3.687500	2.8125
std	NaN	6.026948	1.785922	123.938694	68.562868	0.534679	0.978457	1.786943	0.504016	0.498991	0.737804	1.6152
min	NaN	10.400000	4.000000	71.100000	52.000000	2.760000	1.513000	14.500000	0.000000	0.000000	3.000000	1.0000
25%	NaN	15.425000	4.000000	120.825000	96.500000	3.080000	2.581250	16.892500	0.000000	0.000000	3.000000	2.0000
50%	NaN	19.200000	6.000000	196.300000	123.000000	3.695000	3.325000	17.710000	0.000000	0.000000	4.000000	2.0000
75%	NaN	22.800000	8.000000	326.000000	180.000000	3.920000	3.610000	18.900000	1.000000	1.000000	4.000000	4.0000
max	NaN	33.900000	8.000000	472.000000	335.000000	4.930000	5.424000	22.900000	1.000000	1.000000	5.000000	8.0000

Exploring a DataFrame

- **describe()** function can be applied to a **Serie**
- Returns different information depending on the type of the Serie

```
dataframe.mpg.describe()
```

```
count    32.000000
mean     20.090625
std       6.026948
min      10.400000
25%      15.425000
50%      19.200000
75%      22.800000
max      33.900000
Name: mpg, dtype: float64
```

```
pd.Series(["Value1", "Value2"]).describe()
```

```
count      2
unique     2
top      Value1
freq       1
dtype: object
```

Exploring a DataFrame

- **df.cov()** returns a DataFrame with the result of applying the covariance function in each of the numerical columns (all with all)

```
dataframe.cov()
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
mpg	36.324103	-9.172379	-633.097208	-320.732056	2.195064	-5.116685	4.509149	2.017137	1.803931	2.135685	-5.363105
cyl	-9.172379	3.189516	199.660282	101.931452	-0.668367	1.367371	-1.886855	-0.729839	-0.465726	-0.649194	1.520161
disp	-633.097208	199.660282	15360.799829	6721.158669	-47.064019	107.684204	-96.051681	-44.377621	-36.564012	-50.802621	79.068750
hp	-320.732056	101.931452	6721.158669	4700.866935	-16.451109	44.192661	-86.770081	-24.987903	-8.320565	-6.358871	83.036290
drat	2.195064	-0.668367	-47.064019	-16.451109	0.285881	-0.372721	0.087141	0.118649	0.190151	0.275988	-0.078407
wt	-5.116685	1.367371	107.684204	44.192661	-0.372721	0.957379	-0.305482	-0.273661	-0.338105	-0.421081	0.675790
qsec	4.509149	-1.886855	-96.051681	-86.770081	0.087141	-0.305482	3.193166	0.670565	-0.204960	-0.280403	-1.894113
vs	2.017137	-0.729839	-44.377621	-24.987903	0.118649	-0.273661	0.670565	0.254032	0.042339	0.076613	-0.463710
am	1.803931	-0.465726	-36.564012	-8.320565	0.190151	-0.338105	-0.204960	0.042339	0.248992	0.292339	0.046371
gear	2.135685	-0.649194	-50.802621	-6.358871	0.275988	-0.421081	-0.280403	0.076613	0.292339	0.544355	0.326613
carb	-5.363105	1.520161	79.068750	83.036290	-0.078407	0.675790	-1.894113	-0.463710	0.046371	0.326613	2.608871

Exploring a DataFrame

- **df.corr()** returns a DataFrame with the result of applying the correlation function in each of the numerical columns (all with all)

```
dataframe.corr()
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
mpg	1.000000	-0.852162	-0.847551	-0.776168	0.681172	-0.867659	0.418684	0.664039	0.599832	0.480285	-0.550925
cyl	-0.852162	1.000000	0.902033	0.832447	-0.699938	0.782496	-0.591242	-0.810812	-0.522607	-0.492687	0.526988
disp	-0.847551	0.902033	1.000000	0.790949	-0.710214	0.887980	-0.433698	-0.710416	-0.591227	-0.555569	0.394977
hp	-0.776168	0.832447	0.790949	1.000000	-0.448759	0.658748	-0.708223	-0.723097	-0.243204	-0.125704	0.749812
drat	0.681172	-0.699938	-0.710214	-0.448759	1.000000	-0.712441	0.091205	0.440278	0.712711	0.699610	-0.090790
wt	-0.867659	0.782496	0.887980	0.658748	-0.712441	1.000000	-0.174716	-0.554916	-0.692495	-0.583287	0.427606
qsec	0.418684	-0.591242	-0.433698	-0.708223	0.091205	-0.174716	1.000000	0.744535	-0.229861	-0.212682	-0.656249
vs	0.664039	-0.810812	-0.710416	-0.723097	0.440278	-0.554916	0.744535	1.000000	0.168345	0.206023	-0.569607
am	0.599832	-0.522607	-0.591227	-0.243204	0.712711	-0.692495	-0.229861	0.168345	1.000000	0.794059	0.057534
gear	0.480285	-0.492687	-0.555569	-0.125704	0.699610	-0.583287	-0.212682	0.206023	0.794059	1.000000	0.274073
carb	-0.550925	0.526988	0.394977	0.749812	-0.090790	0.427606	-0.656249	-0.569607	0.057534	0.274073	1.000000

Exploring a DataFrame

- Using the Python visualization tools we could visualize the correlation matrix

```
corr = dataframe.corr()  
corr.style.background_gradient(cmap='coolwarm')
```

	mpg	cyl	displacement	hp	drat	wt	qsec	vs	am	gear	carb
mpg	1.000000	-0.852162	-0.847551	-0.776168	0.681172	-0.867659	0.418684	0.664039	0.599832	0.480285	-0.550925
cyl	-0.852162	1.000000	0.902033	0.832447	-0.699938	0.782496	-0.591242	-0.810812	-0.522607	-0.492687	0.526988
displacement	-0.847551	0.902033	1.000000	0.790949	-0.710214	0.887980	-0.433698	-0.710416	-0.591227	-0.555569	0.394977
hp	-0.776168	0.832447	0.790949	1.000000	-0.448759	0.658748	-0.708223	-0.723097	-0.243204	-0.125704	0.749812
drat	0.681172	-0.699938	-0.710214	-0.448759	1.000000	-0.712441	0.091205	0.440278	0.712711	0.699610	-0.090790
wt	-0.867659	0.782496	0.887980	0.658748	-0.712441	1.000000	-0.174716	-0.554916	-0.692495	-0.583287	0.427606
qsec	0.418684	-0.591242	-0.433698	-0.708223	0.091205	-0.174716	1.000000	0.744535	-0.229861	-0.212682	-0.656249
vs	0.664039	-0.810812	-0.710416	-0.723097	0.440278	-0.554916	0.744535	1.000000	0.168345	0.206023	-0.569607
am	0.599832	-0.522607	-0.591227	-0.243204	0.712711	-0.692495	-0.229861	0.168345	1.000000	0.794059	0.057534
gear	0.480285	-0.492687	-0.555569	-0.125704	0.699610	-0.583287	-0.212682	0.206023	0.794059	1.000000	0.274073
carb	-0.550925	0.526988	0.394977	0.749812	-0.090790	0.427606	-0.656249	-0.569607	0.057534	0.274073	1.000000

Exploring a DataFrame

- **df.head()** returns the first rows of the DataFrame
- We can specify the number of rows

```
dataframe.head()
```

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

```
dataframe.head(1)
```

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.9	2.62	16.46	0	1	4	4

Exploring a DataFrame

- **df.head()** is similar to the **LIMIT** clause in SQL

```
select *  
from tabla  
limit 5
```



Exploring a DataFrame

- **df.tail()** returns the last rows of the DataFrame
- We can specify the number of rows

```
dataframe.tail()
```

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
27	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9	1	1	5	2
28	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.5	0	1	5	4
29	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.5	0	1	5	6
30	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.6	0	1	5	8
31	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.6	1	1	4	2

```
dataframe.tail(1)
```

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
31	Volvo 142E	21.4	4	121.0	109	4.11	2.78	18.6	1	1	4	2



Obtain a data sample

- **df.sample()** returns a sample of the Dataframe

```
dataframe.sample(2)
```

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
11	Merc 450SE	16.4	8	275.8	180	3.07	4.07	17.40	0	0	3	3
5	Valiant	18.1	6	225.0	105	2.76	3.46	20.22	1	0	3	1

Obtain a data sample

- If we want always the same sample we have to set the **random_state** parameter

```
dataframe.sample(2, random_state= 1000)
```

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
6	Duster 360	14.3	8	360.0	245	3.21	3.57	15.84	0	0	3	4
26	Porsche 914-2	26.0	4	120.3	91	4.43	2.14	16.70	0	1	5	2

```
dataframe.sample(2, random_state= 1000)
```

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
6	Duster 360	14.3	8	360.0	245	3.21	3.57	15.84	0	0	3	4
26	Porsche 914-2	26.0	4	120.3	91	4.43	2.14	16.70	0	1	5	2

Exercise 4 (1/2)

- Load the league data that is in an Excel file called "LigaBBVA_20170329.xlsx" and load it into a variable called 'liga'

	#	Equipo	PJ	V	E	D	GF	GC	PTS
0	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0
3	3	Sevilla	28.0	17.0	6.0	5.0	52.0	34.0	57.0
4	4	Atlético Madrid	28.0	16.0	7.0	5.0	52.0	23.0	55.0
5	5	Villarreal	28.0	13.0	9.0	6.0	39.0	20.0	48.0
6	6	Real Sociedad	28.0	15.0	3.0	10.0	42.0	39.0	48.0
7	7	Ath. Bilbao	28.0	13.0	5.0	10.0	35.0	32.0	44.0
8	8	Eibar	28.0	11.0	8.0	9.0	44.0	39.0	41.0
9	9	RCO Espanol	28.0	10.0	10.0	8.0	40.0	30.0	40.0

Exercise 4 (2/2)

- Display summary information about the Dataframe
- Describe all the dataset's variables
- Show the first 3 rows
- Show the last 2 rows
- Show a sample of 4 rows
- Check if any of the columns correlate with any other
- Show the number of values in each column
- Display statistical information about the column 'V'

	count	mean	std	min	25%	50%	75%	max
V	20.0	10.4	5.164657	1.0	6.0	10.0	13.5	20.0



Common operations in a Dataframe

- Pandas has a set of methods to perform lots of operations over a dataset to analyze and manipulate the data
- All these functions have in common is that they **do not modify** the dataframe, but return another dataframe
- This allows concatenate operations until the required result is achieved



Agenda

- Reading / Writing data
- Exploring a DataFrame
- **Renaming headings**
- Filtering Columns
- Filtering Rows
- Sorting Data
- Changing Data
- Deleting Data
- Grouping Data
- Concatenating Data
- Joining Data
- Pivot Tables



Rename columns

- This operation lets us **rename columns** in our dataset

```
select columna as "NewName"  
from tabla
```



Rename columns

- The **rename()** method allows you to rename the column index of a Dataframe

```
dataframe = dataframe.rename(columns = {'C1' : 'Country',  
                                       'C2' : 'Year',  
                                       'C3' : 'Cases',  
                                       'C4' : 'Population'})  
dataframe
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

	C1	C2	C3	C4
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Rename columns

- We could use a **lambda function** to make a transformation in the actual column names

```
[6] dataframe.rename(columns = lambda column : column.lower())
```

	country	year	cases	population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583



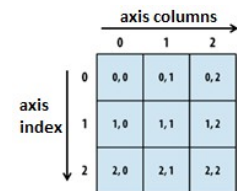
Rename columns

- Another option is use **set_axis()** method to rename the dataset with a list of columns

```
dataframe.set_axis(['Country', 'Year', 'Cases', 'Population'],  
                    axis = 'columns')
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

	C1	C2	C3	C4
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583



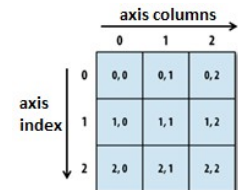
Rename columns

- There are certain cases where the header is a row from the dataset itself
- We have to drop the row with the header after the dataset is renamed

```
(dataframe  
    .set_axis(dataframe.iloc[0], axis = 'columns')  
    .iloc[1:]  
)
```

	Country	Year	Cases	Population
1	Afghanistan	1999	745	19987071
2	Afghanistan	2000	2666	20595360
3	Brazil	1999	37737	172006362
4	Brazil	2000	80488	174504898
5	China	1999	212258	1272815272
6	China	2000	213766	1280428583

	C1	C2	C3	C4
0	Country	Year	Cases	Population
1	Afghanistan	1999	745	19987071
2	Afghanistan	2000	2666	20595360
3	Brazil	1999	37737	172006362
4	Brazil	2000	80488	174504898
5	China	1999	212258	1272815272
6	China	2000	213766	1280428583



Rename rows

- With the parameter **index** (instead of columns) you can rename the dataframe's row index

```
dataframe.rename(index = lambda x : 'Row ' + str(x))
```

	Country	Year	Cases	Population
Row 0	Afghanistan	1999	745	19987071
Row 1	Afghanistan	2000	2666	20595360
Row 2	Brazil	1999	37737	172006362
Row 3	Brazil	2000	80488	174504898
Row 4	China	1999	212258	1272915272
Row 5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Exercise 5

- About the DataFrame “liga”:
- Rename the name of the columns making sure they are materialized in the dataset

Antes	Después
#	Puesto
PJ	PartidosJugados
V	Victorias
E	Empates
D	Derrotas
GF	GolesFavor
GC	GolesContra
PTS	Puntos

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos
0	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0
3	3	Sevilla	28.0	17.0	6.0	5.0	52.0	34.0	57.0

Exercise 6

- About the DataFrame “liga”:
- Rename the columns to convert them to capital letters (but do not materialize it in the DataFrame)

	PUESTO	EQUIPO	PARTIDOSJUGADOS	VICTORIAS	EMPATES	DERROTAS	GOLESFAVOR	GOLESCONTRA	PUNTOS
0	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0



Agenda

- Reading / Writing data
- Exploring a DataFrame
- Renaming headings
- **Filtering Columns**
- Filtering Rows
- Sorting Data
- Changing Data
- Deleting Data
- Grouping Data
- Concatenating Data
- Joining Data
- Pivot Tables



Column Selection

- This operation lets us **filter columns** in our dataset

```
SELECT column1, column2  
FROM table
```

Column Selection

- The **filter()** function is used to Subset rows or columns of the dataframe according to labels in the specified index.

```
dataframe.filter(["Country", "Year"])
```

	Country	Year
0	Afghanistan	1999
1	Afghanistan	2000
2	Brazil	1999
3	Brazil	2000
4	China	1999
5	China	2000

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Column Selection

- The **filter()** function can be used to **reorder** the columns ...

```
dataframe.filter(["Cases", "Population", "Country", "Year"])
```

	Cases	Population	Country	Year
0	745	19987071	Afghanistan	1999
1	2666	20595360	Afghanistan	2000
2	37737	172006362	Brazil	1999
3	80488	174504898	Brazil	2000
4	212258	1272815272	China	1999
5	213766	1280428583	China	2000

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583



Column Selection

- The **filter()** function admits a regular expression instead a list of columns

```
dataframe.filter(regex= "^C")
```

	Country	Cases
0	Afghanistan	745
1	Afghanistan	2666
2	Brazil	37737
3	Brazil	80488
4	China	212258
5	China	213766

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Selecting unique values

- This operation lets us select **distinct values** in the columns of our dataset

```
select distinct column1, column2  
from table
```

Selecting unique values

- To select unique values we have to use the **df.drop_duplicates()** function

```
dataframe.filter(["Country"]) \
    .drop_duplicates()
```

	Country
0	Afghanistan
2	Brazil
4	China

dataframe

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	745	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

```
select distinct Country
from dataset
```

Selecting unique values

- We can select one or more columns

```
dataframe.filter(["Year", "Cases"]) \
    .drop_duplicates() \
    .sort_values("Year")
```

	Year	Cases
0	1999	745
4	1999	212258
1	2000	2666
3	2000	80488
5	2000	213766

dataframe

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	745	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

```
select distinct Year, Cases
from dataset
```



Exercise 7 (1/2)

- About the DataFrame "liga":
- Select the "Puesto" and "Puntos" columns. Limit to 3 rows

	Puesto	Puntos
0	1	65.0
1	2	63.0
2	3	57.0

- Select all the columns with "Puesto" and "Puntos" (5 rows) using a regex

	PartidosJugados	GolesFavor	GolesContra	DiferenciaGoles	PorcentajeGoles
0	27.0	71.0	28.0	43.0	0.087871
1	28.0	81.0	25.0	56.0	0.100248
2	28.0	52.0	34.0	18.0	0.064356
3	28.0	52.0	23.0	29.0	0.064356
4	28.0	39.0	20.0	19.0	0.048267

Exercise 7 (1/2)

- Select unique values of “PartidosJugados” and “Empates” columns

	PartidosJugados	Empates
0	27.0	5.0
1	28.0	6.0
3	28.0	7.0
4	28.0	9.0
5	28.0	3.0
6	28.0	5.0
7	28.0	8.0
8	28.0	10.0

- Select all columns in alphabetical order (sorted)

	Derrotas	Empates	Equipo	GolesContra	GolesFavor	PartidosJugados	Puesto	Puntos	Victorias
0	NaN	NaN	NaN	NaN	NaN	NaN	0	NaN	NaN
1	2.0	5.0	Real Madrid	28.0	71.0	27.0	1	65.0	20.0
2	3.0	6.0	Barcelona	25.0	81.0	28.0	2	63.0	10.0

Agenda

- Reading / Writing data
- Exploring a DataFrame
- Renaming headings
- Filtering Columns
- **Filtering Rows**
- Sorting Data
- Changing Data
- Deleting Data
- Grouping Data
- Concatenating Data
- Joining Data
- Pivot Tables



Row filtering

- This operation lets us **filter rows** in our dataset

```
select *  
from tabla  
where columna = 'A'
```

Row filtering

IMPORTANT

- The **query()** function allows you to filter the rows of a Dataframe in the same way that the **WHERE** clause in a **SQL** statement does

```
dataframe.query("Country == 'China' or Year == 1999")
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
2	Brazil	1999	37737	172006362
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.query("Year not in (2000, 2001)")
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
2	Brazil	1999	37737	172006362
4	China	1999	212258	1272915272



Row filtering

- Note that you have to use a backslash (\) with multiline queries

```
dataframe.query("""Country == 'China' \
or Year == 1999""")
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
2	Brazil	1999	37737	172006362
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Row filtering

- The **query()** function don't work with NULL expresions (at this very moment) but you can use **isnull()** or **notnull()** function

```
dataframe.query("Year.is_not.NULL")
```

```
-----  
KeyError                                Traceback (most recent call last)  
/usr/local/lib/python3.6/dist-packages/pandas/core/computation/scope.py in resolve(self, key, is_local)  
    187         if self.has_resolvers:  
--> 188             return self.resolvers[key]  
    189
```

⬆ 22 frames

```
KeyError: 'NULL'
```

During handling of the above exception, another exception occurred:

```
KeyError                                Traceback (most recent call last)  
KeyError: 'NULL'
```

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

```
UndefinedVariableError                  Traceback (most recent call last)  
/usr/local/lib/python3.6/dist-packages/pandas/core/computation/scope.py in resolve(self, key, is_local)  
    202         from pandas.core.computation.ops import UndefinedVariableError  
    203  
--> 204         raise UndefinedVariableError(key, is_local) from err  
    205  
    206     def swapkey(self, old_key: str, new_key: str, new_value=None):
```

```
UndefinedVariableError: name 'NULL' is not defined
```

SEARCH STACK OVERFLOW

Row filtering

- You can include any Pandas function in a query
- For older versions of Pandas (< 1.2) the parameter `engine="python"` is required.

```
dataframe.query("""Year == 1999 \
and Country.isin(['China']) \
and Population.between(0, 3272815272) \
and Cases.notnull()""")
```

	Country	Year	Cases	Population
4	China	1999-01-01	212258	1272815272

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

String Columns

- Pandas provides lots of vectorized string functions for string columns
- All these functions are available through the **str** property

```
dataframe.query("""Country.str.contains('^C') \\  
and Country.str.len() == 5""")
```

	Country	Year	Cases	Population
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Date Columns

- It is possible to compare a date column with a string literal with a specific format ('YYYY-MM-DD' or 'YYYY-MM-DD HH:MM:DD')
- A datetime column has the **dt** property to access individual components of a date

```
dataframe.query("""Year >= '1999-06-01 04:33:22' \
and Year == '2000-01-01' \
and Year.dt.month == 1 \
and Year.between('2000-01-01', '2003-03-01')""")
```

	Country	Year	Cases	Population
1	Afghanistan	2000-01-01	2666	20595360
3	Brazil	2000-01-01	80488	174504898
5	China	2000-01-01	213766	1280428583

dataframe

	Country	Year	Cases	Population
0	Afghanistan	1999-01-01	745	19987071
1	Afghanistan	2000-01-01	2666	20595360
2	Brazil	1999-01-01	37737	172006362
3	Brazil	2000-01-01	80488	174504898
4	China	1999-01-01	212258	1272815272
5	China	2000-01-01	213766	1280428583

Row filtering

- Pandas lets you filter rows specifying a **boolean array**

```
dataframe[dataframe.Country == "Brazil"]
```

	Country	Year	Cases	Population
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Slices

- We could select a subset of rows specifying a **slice** (two integer numbers separated by a colon)

```
dataframe[4:]
```

	Country	Year	Cases	Population
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

```
dataframe.iloc[4:]
```

	Country	Year	Cases	Population
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Exercise 8

- About the DataFrame “liga”:
- Search rows for Real Madrid and Barcelona

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0

- Search rows whose ‘Puesto’ is less than or equal to 2 or more than or equal to 20

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos
0	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0
20	20	Osasuna	28.0	1.0	8.0	19.0	28.0	67.0	11.0

Exercise 9

- Search rows with the 'Equipo' field is NULL

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos
0	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

- Search rows whose 'Victorias' are greater than or equal to 18 and the 'Goles Favor' are greater than 60

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0

Agenda

- Reading / Writing data
- Exploring a DataFrame
- Renaming headings
- Filtering Columns
- Filtering Rows
- **Sorting Data**
- Changing Data
- Deleting Data
- Grouping Data
- Concatenating Data
- Joining Data
- Pivot Tables



Ordering rows

- This operation lets us **sort rows** in our dataset

```
SELECT column1, column2  
FROM table  
ORDER BY column1
```



Ordering rows

- The **sort_values()** function allows you to **sort** through the values of the DataFrame

```
dataframe.sort_values("Population")
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.sort_values(["Country", "Year"])
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Ordering rows

- The **ascending** parameter allows you to set the ascending / descending order

```
dataframe.sort_values(["Year", "Country"],  
                      ascending = [False, True])
```

	Country	Year	Cases	Population
1	Afghanistan	2000	2666	20595360
3	Brazil	2000	80488	174504898
5	China	2000	213766	1280428583
0	Afghanistan	1999	745	19987071
2	Brazil	1999	37737	172006362
4	China	1999	212258	1272915272

```
dataframe.sort_values(["Country", "Year"],  
                      ascending = [False, False])
```

	Country	Year	Cases	Population
5	China	2000	213766	1280428583
4	China	1999	212258	1272915272
3	Brazil	2000	80488	174504898
2	Brazil	1999	37737	172006362
1	Afghanistan	2000	2666	20595360
0	Afghanistan	1999	745	19987071

Ordering indexes

- The sorting of the index (rows or columns) is done through the **sort_index()** function

```
dataframe.sort_index(ascending = False)
```

	Country	Year	Cases	Population
5	China	2000	213766	1280428583
4	China	1999	212258	1272915272
3	Brazil	2000	80488	174504898
2	Brazil	1999	37737	172006362
1	Afghanistan	2000	2666	20595360
0	Afghanistan	1999	745	19987071

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

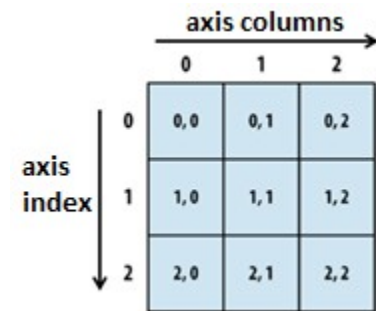
Ordering indexes

- With the **axis** parameter we can sort the columns instead of the rows

```
dataframe.sort_index(ascending=False, axis = 'columns')
```

	Year	Population	Country	Cases
0	1999	19987071	Afghanistan	745
1	2000	20595360	Afghanistan	2666
2	1999	172006362	Brazil	37737
3	2000	174504898	Brazil	80488
4	1999	1272815272	China	212258
5	2000	1280428583	China	213766

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583



Exercise 10 (1/2)

- About the DataFrame "league":
- Sort the DataFrame by index (Descending)

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos
20	20	Osasuna	28.0	1.0	8.0	19.0	28.0	67.0	11.0
19	19	Granada	28.0	4.0	7.0	17.0	25.0	58.0	19.0

- Sort the DataFrame by the 'Puesto' column (Descending)

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos
20	20	Osasuna	28.0	1.0	8.0	19.0	28.0	67.0	11.0
19	19	Granada	28.0	4.0	7.0	17.0	25.0	58.0	19.0



Exercise 10 (2/2)

- Sort the DataFrame by the 'PartidosJugados' (Ascending), 'Victorias' (Descending) and 'GolesFavor' (Ascending) columns

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0
11	11	Celta de Vigo	27.0	11.0	5.0	11.0	40.0	45.0	38.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0
3	3	Sevilla	28.0	17.0	6.0	5.0	52.0	34.0	57.0
4	4	Atlético Madrid	28.0	16.0	7.0	5.0	52.0	23.0	55.0

Agenda

- Reading / Writing data
- Exploring a DataFrame
- Renaming headings
- Filtering Columns
- Filtering Rows
- Sorting Data
- **Changing Data**
- Deleting Data
- Grouping Data
- Concatenating Data
- Joining Data
- Pivot Tables



Change columns, or add new ones

- This operation lets us **add** new columns or **change** existings ones in our dataset

```
SELECT new_column as column1 + column2  
FROM table
```



Change columns, or add new ones

- In Pandas we can create new columns by assigning a value directly to the new column
- This operation materialize the new column in the dataset

```
dataframe["Id"] = range(len(dataframe))  
dataframe["Id1"] = 1  
dataframe
```

	Country	Year	Cases	Population	Id	Id1
0	Afghanistan	1999	745	19987071	0	1
1	Afghanistan	2000	2666	20595360	1	1
2	Brazil	1999	37737	172006362	2	1
3	Brazil	2000	80488	174504898	3	1
4	China	1999	212258	1272815272	4	1
5	China	2000	213766	1280428583	5	1

dataframe

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

Change columns, or add new ones

- The **assign()** function makes easier adding new columns to our dataset

```
SELECT column1, column2,  
       column1 + column2 as new_column  
FROM table
```

```
dataframe.assign(Id = range(len(dataframe)),  
                 Id1 = 1)
```

	Country	Year	Cases	Population	Id	Id1
0	Afghanistan	1999	745	19987071	0	1
1	Afghanistan	2000	2666	20595360	1	1
2	Brazil	1999	37737	172006362	2	1
3	Brazil	2000	80488	174504898	3	1
4	China	1999	212258	1272815272	4	1
5	China	2000	213766	1280428583	5	1

dataframe

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

Change columns, or add new ones

- It even allows us to change existing columns in our dataset

```
dataframe.assign(Year = dataframe.Year + 100)
```

	Country	Year	Cases	Population	Id	Id1
0	Afghanistan	2099	745	19987071	0	1
1	Afghanistan	2100	2666	20595360	1	1
2	Brazil	2099	37737	172006362	2	1
3	Brazil	2100	80488	174504898	3	1
4	China	2099	212258	1272815272	4	1
5	China	2100	213766	1280428583	5	1

dataframe

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

```
dataframe.Year + 100
```

```
0    2099
1    2100
2    2099
3    2100
4    2099
5    2100
Name: Year, dtype: int64
```


Change columns, or add new ones

- We can use a **lambda** function or anything that returns a Serie

```
dataframe.assign(  
    CasesAcum = dataframe.Cases.cumsum(),  
    CasesPercent = lambda row : row.Cases / row.Cases.sum()  
)
```

	Country	Year	Cases	Population	Id	Id1	CasesAcum	CasesPercent
0	Afghanistan	1999	745	19987071	0	1	745	0.001360
1	Afghanistan	2000	2666	20595360	1	1	3411	0.004868
2	Brazil	1999	37737	172006362	2	1	41148	0.068906
3	Brazil	2000	80488	174504898	3	1	121636	0.146967
4	China	1999	212258	1272815272	4	1	333894	0.387573
5	China	2000	213766	1280428583	5	1	547660	0.390326

dataframe

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

dataframe.Cases.cumsum()

```
0    745  
1   3411  
2   41148  
3  121636  
4  333894  
5  547660  
Name: Cases, dtype: int64
```

Change columns, or add new ones

- We can **only** use vectorized functions

```
dataframe.assign(  
    CountryHash = hash(dataframe.Country)  
)
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-61-0b6c96b7446e> in <module>  
      1 dataframe.assign(  
----> 2     CountryHash = hash(dataframe.Country)  
      3 )  
  
c:\users\daniel\.virtualenvs\practicass-pandas-cjitosop\lib\site-packages\pan  
1783  
1784     def __hash__(self) -> int:  
-> 1785         raise TypeError(  
1786             f"{repr(type(self).__name__)} objects are mutable, "  
1787             f"thus they cannot be hashed"  
  
TypeError: 'Series' objects are mutable, thus they cannot be hashed
```

dataframe				
	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

```
print(hash("Afghanistan"))  
-1794632447089383926
```

Change columns, or add new ones

- **np.vectorize()** lets us to convert any python function into a vectorized one

```
hashed = np.vectorize(lambda x: hash(x))
dataframe.assign(
    CountryHash = hashed(dataframe.Country)
)
```

	Country	Year	Cases	Population	Id	Id1	CountryHash
0	Afghanistan	1999	745	19987071	0	1	7416641866878524706
1	Afghanistan	2000	2666	20595360	1	1	7416641866878524706
2	Brazil	1999	37737	172006362	2	1	-5670732118295316485
3	Brazil	2000	80488	174504898	3	1	-5670732118295316485
4	China	1999	212258	1272815272	4	1	9070864174946750906
5	China	2000	213766	1280428583	5	1	9070864174946750906

dataframe

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

```
print(hash("Afghanistan"))
-1794632447089383926
```

Binning data

- Pandas provides the **cut()** function to bin a continuous variable into discrete intervals

```
n = [61, 16, 21, 62, 80, 55, 32, 20, 53, 22]
n = sorted(n)
```

```
pd.DataFrame( {
    'N' : n,
    'Cut 3 Bins': pd.cut(n, bins = 3),
    'Cut 3 Label': pd.cut(n, bins = 3, labels = ['Group 1', 'Group 2', 'Group 3']),
    'Cut 3 Vector': pd.cut(n, bins = [10,32,70, 100], labels = ['10-32', '33-70', '71-100'])
})
```

	N	Cut 3 Bins	Cut 3 Label	Cut 3 Vector
0	16	(15.936, 37.333]	Group 1	10-32
1	20	(15.936, 37.333]	Group 1	10-32
2	21	(15.936, 37.333]	Group 1	10-32
3	22	(15.936, 37.333]	Group 1	10-32
4	32	(15.936, 37.333]	Group 1	10-32
5	53	(37.333, 58.667]	Group 2	33-70
6	55	(37.333, 58.667]	Group 2	33-70
7	61	(58.667, 80.0]	Group 3	33-70
8	62	(58.667, 80.0]	Group 3	33-70
9	80	(58.667, 80.0]	Group 3	71-100

Binning data

- We can use this function to group a value into a range

```
dataframe.assign(  
    Type = pd.cut(dataframe.Cases,  
                  bins = [0, 50000, 5000000],  
                  labels= ["Type A", "Type B"])  
)
```

	Country	Year	Cases	Population	Type
0	Afghanistan	1999	745	19987071	Type A
1	Afghanistan	2000	2666	20595360	Type A
2	Brazil	1999	37737	172006362	Type A
3	Brazil	2000	80488	174504898	Type B
4	China	1999	212258	1272815272	Type B
5	China	2000	213766	1280428583	Type B

dataframe				
	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

Calculating Trends

- Calculating trends is common when we have a time serie

```
np.random.seed(seed=100)

time_series = pd.DataFrame(
    {
        'Date' : pd.date_range(start = '2020-01-01', end = '2020-01-05', freq = 'D'),
        'Value' : np.random.randint(100, size = 5)
    }
)
```

time_series

	Date	Value
0	2020-01-01	8
1	2020-01-02	24
2	2020-01-03	67
3	2020-01-04	87
4	2020-01-05	79

Calculating Trends

- The **shift()** function calculates the previous (or subsequent) value in the time series

```
time_series.assign(  
    PreviousValue = lambda row : row.Value.shift(1),  
    Increment = lambda row : row.Value - row.Value.shift(1),  
)
```

	Date	Value	PreviousValue	Increment
0	2020-01-01	8	NaN	NaN
1	2020-01-02	24	8.0	16.0
2	2020-01-03	67	24.0	43.0
3	2020-01-04	87	67.0	20.0
4	2020-01-05	79	87.0	-8.0



Date Columns

- A datetime column has the **dt** property to access individual components of a date

```
df.assign(  
    year = df.DoB.dt.year,  
    month = df.DoB.dt.month,  
    day = df.DoB.dt.day,  
    week = df.DoB.dt.isocalendar().week,  
    dayofweek = df.DoB.dt.dayofweek,  
    is_leap_year = df.DoB.dt.is_leap_year  
)
```

	name	DoB	year	month	day	week	dayofweek	is_leap_year
0	Tom	1997-08-05	1997	8	5	32	1	False
1	Andy	1996-04-28	1996	4	28	17	6	True
2	Lucas	1995-12-16	1995	12	16	50	5	False

df

	name	DoB
0	Tom	1997-08-05
1	Andy	1996-04-28
2	Lucas	1995-12-16



Exercise 11 (1/2)

- On the DataFrame “liga”, create the following columns:
- ‘DiferenciaGoles’= ‘GolesFavor’ minus ‘GolesContra’
- ‘PorcentajeGoles’ = ‘GolesFavor’ / Sum of the ‘GolesFavor’ of all the teams
- PercentageVictorias = Victories / Matches Played
- PointsAcum = Accumulated ‘Puntos’ of all teams (cumsum)
- Materialize this new columns on the dataset



Exercise 11 (2/2)

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos	DiferenciaGoles	PorcentajeGoles	PorcentajeVictorias	PuntosAcum
0	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0	43.0	0.087871	0.740741	65.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0	56.0	0.100248	0.678571	128.0
3	3	Sevilla	28.0	17.0	6.0	5.0	52.0	34.0	57.0	18.0	0.064356	0.607143	185.0
4	4	Atlético Madrid	28.0	16.0	7.0	5.0	52.0	23.0	55.0	29.0	0.064356	0.571429	240.0
5	5	Villarreal	28.0	13.0	9.0	6.0	39.0	20.0	48.0	19.0	0.048267	0.464286	288.0
6	6	Real Sociedad	28.0	15.0	3.0	10.0	42.0	39.0	48.0	3.0	0.051980	0.535714	336.0
7	7	Ath. Bilbao	28.0	13.0	5.0	10.0	35.0	32.0	44.0	3.0	0.043317	0.464286	380.0
8	8	Eibar	28.0	11.0	8.0	9.0	44.0	39.0	41.0	5.0	0.054455	0.392857	421.0
9	9	RCD Espanyol	28.0	10.0	10.0	8.0	40.0	39.0	40.0	1.0	0.049505	0.357143	461.0
10	10	Alavés	28.0	10.0	10.0	8.0	29.0	33.0	40.0	-4.0	0.035891	0.357143	501.0
11	11	Celta de Vigo	27.0	11.0	5.0	11.0	40.0	45.0	38.0	-5.0	0.049505	0.407407	539.0
12	12	U. D. Las Palmas	28.0	9.0	8.0	11.0	44.0	45.0	35.0	-1.0	0.054455	0.321429	574.0
13	13	Betis	28.0	8.0	7.0	13.0	31.0	44.0	31.0	-13.0	0.038366	0.285714	605.0
14	14	Valencia C. F.	28.0	8.0	6.0	14.0	38.0	51.0	30.0	-13.0	0.047030	0.285714	635.0
15	15	Málaga	28.0	6.0	9.0	13.0	33.0	45.0	27.0	-12.0	0.040842	0.214286	662.0
16	16	Deportivo	28.0	6.0	9.0	13.0	31.0	43.0	27.0	-12.0	0.038366	0.214286	689.0
17	17	Leganés	28.0	6.0	8.0	14.0	22.0	41.0	26.0	-19.0	0.027228	0.214286	715.0
18	18	Sporting Gijón	28.0	5.0	6.0	17.0	31.0	57.0	21.0	-26.0	0.038366	0.178571	736.0
19	19	Granada	28.0	4.0	7.0	17.0	25.0	58.0	19.0	-33.0	0.030941	0.142857	755.0
20	20	Osasuna	28.0	1.0	8.0	19.0	28.0	67.0	11.0	-39.0	0.034653	0.035714	766.0

Exercise 12 (1/2)

- On the DataFrame "liga", create the following columns:
- Zona = "Champions" if the team is in one of the first 4 places, "Descenso" if the team is in the last 3 places, "Normal" for the rest of the cases (pd.cut)
- DiferenciaPuntos = Difference in 'Puntos' between a team and the team immediately below it in the ranking (series.shift(n))
- Temporada = "2016-2017"
- Materialize this new columns on the dataset



Exercise 12 (2/2)

	Puesto	Equipo	Puntos	Zona	DiferenciaPuntos	Temporada
0	0	NaN	NaN	NaN	NaN	2016-2017
1	1	Real Madrid	65.0	Champions	2.0	2016-2017
2	2	Barcelona	63.0	Champions	6.0	2016-2017
3	3	Sevilla	57.0	Champions	2.0	2016-2017
4	4	Atlético Madrid	55.0	Champions	7.0	2016-2017
5	5	Villarreal	48.0	Normal	0.0	2016-2017
6	6	Real Sociedad	48.0	Normal	4.0	2016-2017
7	7	Ath. Bilbao	44.0	Normal	3.0	2016-2017
8	8	Eibar	41.0	Normal	1.0	2016-2017
9	9	RCD Espanyol	40.0	Normal	0.0	2016-2017
10	10	Alavés	40.0	Normal	2.0	2016-2017
11	11	Celta de Vigo	38.0	Normal	3.0	2016-2017
12	12	U. D. Las Palmas	35.0	Normal	4.0	2016-2017
13	13	Betis	31.0	Normal	1.0	2016-2017
14	14	Valencia C. F.	30.0	Normal	3.0	2016-2017
15	15	Málaga	27.0	Normal	0.0	2016-2017
16	16	Deportivo	27.0	Normal	1.0	2016-2017
17	17	Leganés	26.0	Normal	5.0	2016-2017
18	18	Sporting Gijón	21.0	Descenso	2.0	2016-2017
19	19	Granada	19.0	Descenso	8.0	2016-2017
20	20	Osasuna	11.0	Descenso	NaN	2016-2017

Agenda

- Reading / Writing data
- Exploring a DataFrame
- Renaming headings
- Filtering Columns
- Filtering Rows
- Sorting Data
- Changing Data
- **Deleting Data**
- Grouping Data
- Concatenating Data
- Joining Data
- Pivot Tables



Deleting rows

- This operation lets us **delete rows** in our dataset

```
DELETE FROM table  
WHERE column1 = 'A'
```

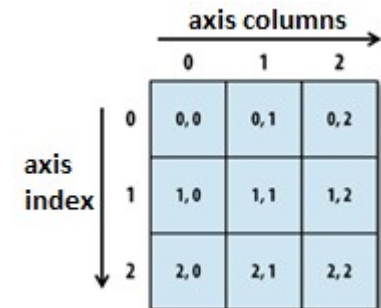
Deleting rows

- The **drop()** function allows you to delete both rows and columns by specifying an array of index names
- It delete rows by default

```
dataframe.drop([1,3])
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
2	Brazil	1999	37737	172006362
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583



Deleting rows

- We **can not** use a condition to delete rows with **drop()**

```
dataframe.drop(dataframe.Country != 'China')
```

```
-----  
KeyError                                Traceback (most recent call last)  
<ipython-input-13-3d4fb419e98e> in <module>  
----> 1 dataframe.drop(dataframe.Country != 'China')  
  
c:\users\daniel\.virtualenvs\practicapandas-cjitosop\lib\site-packages\pandas\core\indexes\base.py in drop(self, labels, errors)  
    5589         if mask.any():  
    5590             if errors != "ignore":  
-> 5591                 raise KeyError(f"{labels[mask]} not found in axis")  
    5592             indexer = indexer[~mask]  
    5593             return self.delete(indexer)  
  
KeyError: '[ True  True  True  True False False] not found in axis'
```


Deleting rows

- An alternative to drop rows with a condition is selecting only the rows that we want to keep using then **query()** function

```
dataframe = dataframe.query("Country != 'China'")  
dataframe
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Deleting rows

- Or we can delete rows selecting the rows we want to keep using the **iloc** property ...

```
dataframe = dataframe.iloc[1:]  
dataframe
```

	Country	Year	Cases	Population
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Deleting columns

```
dataframe.drop(columns = 'Year')
```

	Country	Cases	Population
0	Afghanistan	745	19987071
1	Afghanistan	2666	20595360
2	Brazil	37737	172006362
3	Brazil	80488	174504898
4	China	212258	1272815272
5	China	213766	1280428583

```
dataframe.drop(columns = ['Country', 'Year'])
```

	Cases	Population
0	745	19987071
1	2666	20595360
2	37737	172006362
3	80488	174504898
4	212258	1272815272
5	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Deleting columns

- An alternative is to select only with the columns that we want to keep in the dataset using the **filter()** function

```
dataframe.filter(['Cases', 'Population'])
```

	Cases	Population
0	745	19987071
1	2666	20595360
2	37737	172006362
3	80488	174504898
4	212258	1272915272
5	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Exercise 13 (1/2)

- About the DataFrame “league”
- Delete row 0
- Delete the “Temporada” column
- Materialize this changes on the dataset



Exercise 13 (2/2)

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos	DiferenciaGoles	PorcentajeGoles	PorcentajeVictorias	PuntosAcum	Zona	DiferenciaPuntos
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0	43.0	0.087871	0.740741	65.0	Champions	2.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0	56.0	0.100248	0.678571	128.0	Champions	6.0
3	3	Sevilla	28.0	17.0	6.0	5.0	52.0	34.0	57.0	18.0	0.064356	0.607143	185.0	Champions	2.0
4	4	Atlético Madrid	28.0	16.0	7.0	5.0	52.0	23.0	55.0	29.0	0.064356	0.571429	240.0	Champions	7.0
5	5	Villarreal	28.0	13.0	9.0	6.0	39.0	20.0	48.0	19.0	0.048267	0.464286	288.0	Normal	0.0
6	6	Real Sociedad	28.0	15.0	3.0	10.0	42.0	39.0	48.0	3.0	0.051980	0.535714	336.0	Normal	4.0
7	7	Ath. Bilbao	28.0	13.0	5.0	10.0	35.0	32.0	44.0	3.0	0.043317	0.464286	380.0	Normal	3.0
8	8	Eibar	28.0	11.0	8.0	9.0	44.0	39.0	41.0	5.0	0.054455	0.392857	421.0	Normal	1.0
9	9	RCD Espanyol	28.0	10.0	10.0	8.0	40.0	39.0	40.0	1.0	0.049505	0.357143	461.0	Normal	0.0
10	10	Alavés	28.0	10.0	10.0	8.0	29.0	33.0	40.0	-4.0	0.035891	0.357143	501.0	Normal	2.0
11	11	Celta de Vigo	27.0	11.0	5.0	11.0	40.0	45.0	38.0	-5.0	0.049505	0.407407	539.0	Normal	3.0
12	12	U. D. Las Palmas	28.0	9.0	8.0	11.0	44.0	45.0	35.0	-1.0	0.054455	0.321429	574.0	Normal	4.0
13	13	Betis	28.0	8.0	7.0	13.0	31.0	44.0	31.0	-13.0	0.038366	0.285714	605.0	Normal	1.0
14	14	Valencia C. F.	28.0	8.0	6.0	14.0	38.0	51.0	30.0	-13.0	0.047030	0.285714	635.0	Normal	3.0
15	15	Málaga	28.0	6.0	9.0	13.0	33.0	45.0	27.0	-12.0	0.040842	0.214286	662.0	Normal	0.0
16	16	Deportivo	28.0	6.0	9.0	13.0	31.0	43.0	27.0	-12.0	0.038366	0.214286	689.0	Normal	1.0
17	17	Leganés	28.0	6.0	8.0	14.0	22.0	41.0	26.0	-19.0	0.027228	0.214286	715.0	Normal	5.0
18	18	Sporting Gijón	28.0	5.0	6.0	17.0	31.0	57.0	21.0	-26.0	0.038366	0.178571	736.0	Descenso	2.0
19	19	Granada	28.0	4.0	7.0	17.0	25.0	58.0	19.0	-33.0	0.030941	0.142857	755.0	Descenso	8.0
20	20	Osasuna	28.0	1.0	8.0	19.0	28.0	67.0	11.0	-39.0	0.034653	0.035714	766.0	Descenso	NaN

Agenda

- Reading / Writing data
- Exploring a DataFrame
- Renaming headings
- Filtering Columns
- Filtering Rows
- Sorting Data
- Changing Data
- Deleting Data
- **Grouping Data**
- Concatenating Data
- Joining Data
- Pivot Tables



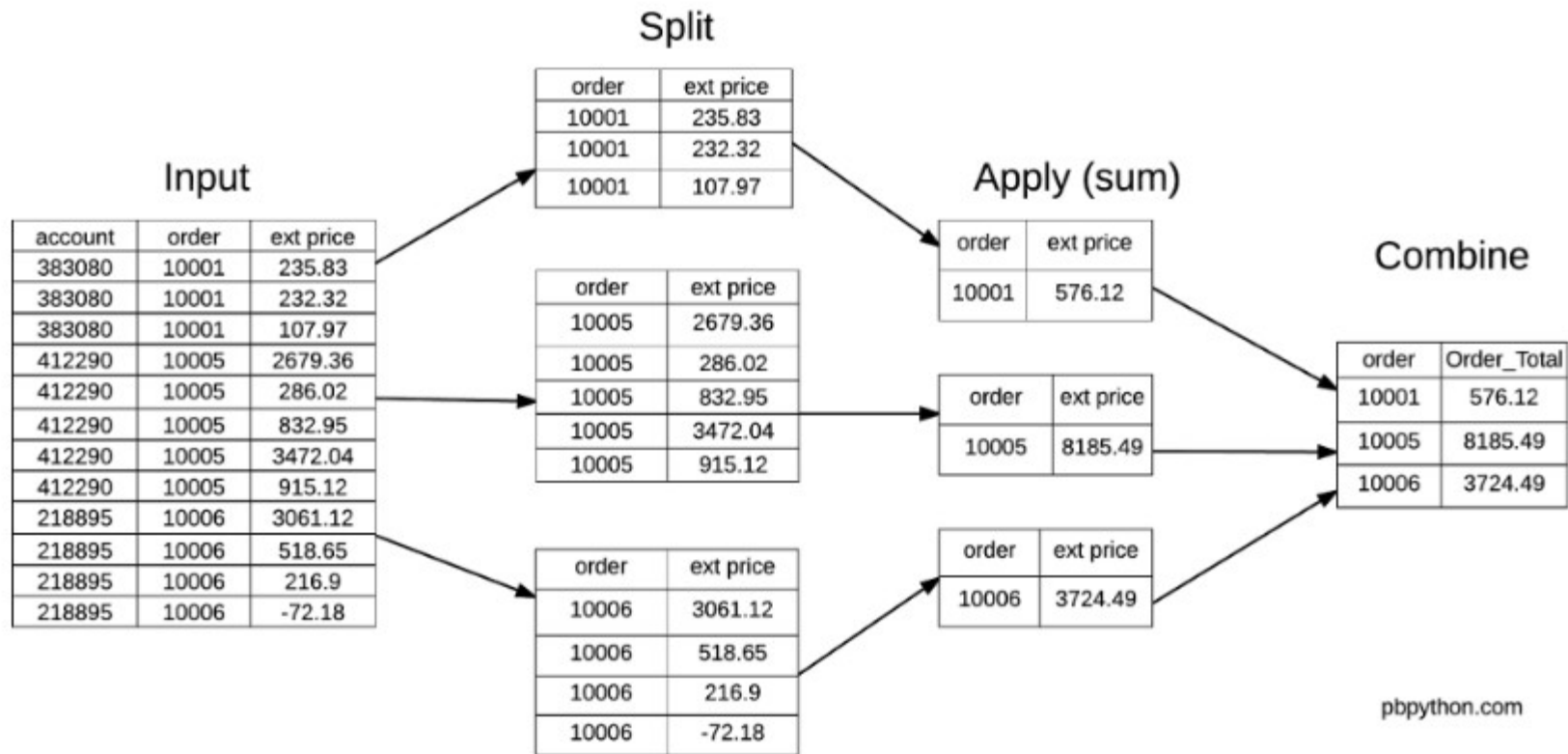
Grouping rows

- This operation lets us **group rows** in our dataset

```
SELECT column1, column2, mean(column3), sum(column4)
FROM some_table
GROUP BY column1, column2
```


Grouping rows

- Pandas has a series of functions that allow to obtain data from groupings



Grouping rows

- The **groupby()** function allows you to group rows and generate groups

```
grouped = dataframe.groupby("Country")
grouped.groups

{'Afghanistan': Int64Index([0, 1], dtype='int64'),
 'Brazil': Int64Index([2, 3], dtype='int64'),
 'China': Int64Index([4, 5], dtype='int64')}
```

```
for name, group in grouped:
    print(name)
    print(group)
```

Afghanistan

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360

Brazil

	Country	Year	Cases	Population
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898

China

	Country	Year	Cases	Population
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Grouping rows

- Once you have a group, you can use grouping functions such as **count()**, **sum()**, **mean()**, **first()**, **last()** etc.

```
dataframe.groupby("Country").count()
```

	Year	Cases	Population
Country			
Afghanistan	2	2	2
Brazil	2	2	2
China	2	2	2

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Grouping rows

[Getting started](#) [User Guide](#) [API reference](#) [Development](#) [Release notes](#)

GroupBy

- [pandas.core.groupby.GroupBy.__iter__](#)
- [pandas.core.groupby.GroupBy.groups](#)
- [pandas.core.groupby.GroupBy.indices](#)
- [pandas.core.groupby.GroupBy.get_group](#)
- [pandas.Grouper](#)
- [pandas.core.groupby.GroupBy.apply](#)
- [pandas.core.groupby.GroupBy.agg](#)
- [pandas.core.groupby.SeriesGroupBy.aggregate](#)
- [pandas.core.groupby.DataFrameGroupBy.agg](#)
- [pandas.core.groupby.SeriesGroupBy.transform](#)
- [pandas.core.groupby.DataFrameGroupBy.transf](#)
- [pandas.core.groupby.GroupBy.pipe](#)
- [pandas.core.groupby.GroupBy.all](#)
- [pandas.core.groupby.GroupBy.any](#)
- [pandas.core.groupby.GroupBy.bfill](#)
- [pandas.core.groupby.GroupBy.backfill](#)
- [pandas.core.groupby.GroupBy.count](#)**
- [pandas.core.groupby.GroupBy.cumcount](#)
- [pandas.core.groupby.GroupBy.cummax](#)
- [pandas.core.groupby.GroupBy.cummin](#)
- [pandas.core.groupby.GroupBy.cumprod](#)
- [pandas.core.groupby.GroupBy.cumsum](#)
- [pandas.core.groupby.GroupBy.ffill](#)
- [pandas.core.groupby.GroupBy.first](#)

pandas.core.groupby.GroupBy.count

[\[source\]](#)**GroupBy.count()**

Compute count of group, excluding missing values.

Returns: Series or DataFrame

Count of values within each group.

See also

[Series.groupby](#)[DataFrame.groupby](#)[pandas.core.groupby.GroupBy.backfill](#)[pandas.core.groupby.GroupBy.cumcount](#)

Grouping rows

```
dataframe.groupby("Country").sum()
```

	Year	Cases	Population
Country			
Afghanistan	3999	3411	40582431
Brazil	3999	118225	346511260
China	3999	426024	2553343855

```
dataframe.groupby("Country").mean()
```

	Year	Cases	Population
Country			
Afghanistan	1999.5	1705.5	2.029122e+07
Brazil	1999.5	59112.5	1.732556e+08
China	1999.5	213012.0	1.276672e+09

```
dataframe.groupby("Country").first()
```

	Year	Cases	Population
Country			
Afghanistan	1999	745	19987071
Brazil	1999	37737	172006362
China	1999	212258	1272915272

```
dataframe.groupby("Country").last()
```

	Year	Cases	Population
Country			
Afghanistan	2000	2666	20595360
Brazil	2000	80488	174504898
China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Grouping rows

- A column is shown depending of its type and the grouping function that we used

```
dataframe.groupby("Cases").mean()
```

	Year	Population
Cases		
745	1999	19987071
2666	2000	20595360
37737	1999	172006362
80488	2000	174504898
212258	1999	1272815272
213766	2000	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Grouping rows

- **size()** counts all rows including null values
- Return a Serie instead a Dataframe

```
dataframe.groupby("Country").size()
```

```
Country  
Afghanistan    2  
Brazil         2  
China         2  
dtype: int64
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Grouping rows

- **size()** allows to check if a group of columns has unique values in the dataset

```
dataframe.groupby(["Country", "Year"]).size()
```

```
Country  Year
Afghanistan  1999    1
             2000    1
Brazil       1999    1
             2000    1
China        1999    1
             2000    1
dtype: int64
```

```
dataframe.groupby(["Country", "Year"]).size() > 1
```

```
Country  Year
Afghanistan  1999    False
             2000    False
Brazil       1999    False
             2000    False
China        1999    False
             2000    False
dtype: bool
```

```
(dataframe.groupby(["Country", "Year"]).size() > 1).sum()
```

0

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Grouping rows by multiple columns

- In case of grouping for more than a column we will use a list of columns as parameter

```
dataframe.groupby(["Country", "Year"]).count()
```

		Cases	Population
Country	Year		
Afghanistan	1999	1	1
	2000	1	1
Brazil	1999	1	1
	2000	1	1
China	1999	1	1
	2000	1	1

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583



Grouping rows

- The **aggregate()** function allows you to apply a set of grouping functions
- Accept a list of functions as parameter

```
dataframe.groupby("Country").aggregate(["mean", "sum"])
```

	Year		Cases		Population	
	mean	sum	mean	sum	mean	sum
Country						
Afghanistan	1999.5	3999	1705.5	3411	20291215	40582431
Brazil	1999.5	3999	59112.5	118225	173255630	346511260
China	1999.5	3999	213012.0	426024	1276671927	2553343855

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Grouping rows

- We can select a **specific column** from the result of the **aggregate()** function.

```
dataframe.groupby("Country").aggregate(["mean", "sum"])
```

	Year		Cases		Population	
	mean	sum	mean	sum	mean	sum
Country						
Afghanistan	1999.5	3999	1705.5	3411	2.029122e+07	40582431
Brazil	1999.5	3999	59112.5	118225	1.732556e+08	346511260
China	1999.5	3999	213012.0	426024	1.276622e+09	2553243855

```
dataframe.groupby("Country").aggregate(["mean", "sum"]).Population
```

	mean	sum
Country		
Afghanistan	2.029122e+07	40582431
Brazil	1.732556e+08	346511260
China	1.276622e+09	2553243855

Grouping rows

- The **agg()** function allows you to obtain new values by applying specific function to a given columns

```
dataframe.groupby("Country").agg({  
    'Year' : "count",  
    "Cases" : np.sum,  
    "Population" : lambda column: column.mean()  
})
```

	Year	Cases	Population
Country			
Afghanistan	2	3411	2.029122e+07
Brazil	2	118225	1.732556e+08
China	2	426024	1.276622e+09

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
SELECT country,  
       count(Year),  
       sum(Cases),  
       mean(Poputation)  
FROM dataframe  
GROUP BY country
```

Transforming rows

- The **transform()** function applies a grouping function to a group, but returns an object with the same size as the original dataframe

```
dataframe.groupby("Country").Cases.sum()
```

```
Country
Afghanistan      3411
Brazil           118225
China            426024
Name: Cases, dtype: int64
```

```
dataframe.groupby("Country").Cases.transform("sum")
```

```
0      3411
1      3411
2     118225
3     118225
4     426024
5     426024
Name: Cases, dtype: int64
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Transforming rows

```
dataframe.assign(  
    TotalCases = dataframe.groupby("Country").Cases.transform("sum")  
)
```

	Country	Year	Cases	Population	TotalCases
0	Afghanistan	1999	745	19987071	3411
1	Afghanistan	2000	2666	20595360	3411
2	Brazil	1999	37737	172006362	118225
3	Brazil	2000	80488	174504898	118225
4	China	1999	212258	1272815272	426024
5	China	2000	213766	1280428583	426024

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583



Reseting the index

- The result of grouping multiple columns is a **DataFrame with a MultiIndex**

```
dataframe.groupby(["Country", "Year"]).count()
```

		Cases	Population
Country	Year		
Afghanistan	1999	1	1
	2000	1	1
Brazil	1999	1	1
	2000	1	1
China	1999	1	1
	2000	1	1

```
dataframe.groupby(["Country", "Year"]).count().index
```

```
MultiIndex([('Afghanistan', 1999),  
            ('Afghanistan', 2000),  
            ('Brazil', 1999),  
            ('Brazil', 2000),  
            ('China', 1999),  
            ('China', 2000)],  
           names=['Country', 'Year'])
```

```
dataframe.groupby(["Country", "Year"]).count().columns
```

```
Index(['Cases', 'Population'], dtype='object')
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Resetting the index

- We can incorporate the index as regular columns with the **reset_index()** function

```
dataframe.groupby(["Country", "Year"]).count()
```

		Cases	Population
Country	Year		
Afghanistan	1999	1	1
	2000	1	1
Brazil	1999	1	1
	2000	1	1
China	1999	1	1
	2000	1	1

```
dataframe.groupby(["Country", "Year"]).count().reset_index()
```

	Country	Year	Cases	Population
0	Afghanistan	1999	1	1
1	Afghanistan	2000	1	1
2	Brazil	1999	1	1
3	Brazil	2000	1	1
4	China	1999	1	1
5	China	2000	1	1

Reseting the index

- Another option is use the **as_index** parameter in the **groupby()** function

```
dataframe.groupby(["Country", "Year"], as_index = False ).count()
```

	Country	Year	Cases	Population	Date_str
0	Afghanistan	1999	1	1	1
1	Afghanistan	2000	1	1	1
2	Brazil	1999	1	1	1
3	Brazil	2000	1	1	1
4	China	1999	1	1	1
5	China	2000	1	1	1

Aggregate all the data

- This operation lets us **group rows** taking into consideration every row of our dataset

```
SELECT mean(column3), sum(column4)  
FROM some_table
```



Aggregate all the data

- In these cases we can use functions like **agg** or **aggregate** without use **groupby** previously

```
dataframe.aggregate("mean")
```

```
Year          1.999500e+03  
Cases         9.127667e+04  
Population    4.900563e+08  
dtype: float64
```

```
dataframe[["Cases", "Population"]].aggregate(["mean", "sum"])
```

	Cases	Population
mean	91276.666667	4.900563e+08
sum	547660.000000	2.940338e+09

```
dataframe.agg({"Cases": "mean", "Population": np.sum})
```

```
Cases          9.127667e+04  
Population     2.940338e+09  
dtype: float64
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Exercise 14

- About the DataFrame "liga"
- Sum "Puntos" and "GolesFavor" for all teams

```
Puntos      766.0  
GolesFavor   808.0  
dtype: float64
```

- By zones, for the fields "Puntos" and "GolesFavor", calculate the sum and count all rows

	Puntos		GolesFavor	
	sum	count	sum	count
Zona				
Champions	240.0	4	256.0	4
Normal	496.0	14	499.0	14
Descenso	30.0	2	53.0	2

Exercise 15

- About the DataFrame "liga"
- Filter the 'Puesto', 'Zona', 'Equipo', and 'Puntos' columns
- Add a new column with the average "Puntos" in each Zone

	Puesto	Zona	Equipo	Puntos	MediaPuntos
1	1	Champions	Real Madrid	65.0	60.000000
2	2	Champions	Barcelona	63.0	60.000000
3	3	Champions	Sevilla	57.0	60.000000
4	4	Champions	Atlético Madrid	55.0	60.000000
5	5	Normal	Villarreal	48.0	36.538462
6	6	Normal	Real Sociedad	48.0	36.538462
7	7	Normal	Ath. Bilbao	44.0	36.538462
8	8	Normal	Eibar	41.0	36.538462



Exercise 16 (1/2)

- By zones:
 - Sum of “PartidosJugados” using a lambda function
 - Sum of “GolesFavor” using a numpy function
 - Average in “Diferencia de Goles”

	PartidosJugados	GolesFavor	DiferenciaGoles
Zona			
Champions	111.0	256.0	36.500000
Normal	363.0	468.0	-3.692308
Descenso	84.0	84.0	-32.666667



Exercise 16 (2/2)

- Add a new column to the previous dataset:
'GolesPartido' : 'GolesFavor' / 'PartidosJugados'

	PartidosJugados	GolesFavor	DiferenciaGoles	GolesPartido
Zona				
Champions	111.0	256.0	36.500000	2.306306
Normal	363.0	468.0	-3.692308	1.289256
Descenso	84.0	84.0	-32.666667	1.000000

- Show all the teams in each zone

	Equipo
Zona	
Champions	Real Madrid / Barcelona / Sevilla / Atlético M...
Normal	Villarreal / Real Sociedad / Ath. Bilbao / Eib...
Descenso	Granada / Osasuna

Agenda

- Reading / Writing data
- Exploring a DataFrame
- Renaming headings
- Filtering Columns
- Filtering Rows
- Sorting Data
- Changing Data
- Deleting Data
- Grouping Data
- **Concatenating Data**
- Joining Data
- Pivot Tables



Concatenating rows

- This operation lets us **concatenate** two datasets

```
SELECT column1, column2,  
FROM table1  
UNION ALL  
SELECT column1, column2  
FROM table2
```

Concatenating rows

- The **concat()** function allows to join several DataFrames in a single one

```
pd.concat((dataframe, dataframe), axis = 'index')
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

		axis columns		
		0	1	2
axis index	0	0,0	0,1	0,2
	1	1,0	1,1	1,2
	2	2,0	2,1	2,2

Concatenating rows

- The **reset_index()** function rebuilds the index after the concatenation ...

```
pd.concat((dataframe, dataframe), axis = 'index') \
.reset_index()
```

	index	Country	Year	Cases	Population
0	0	Afghanistan	1999	745	19987071
1	1	Afghanistan	2000	2666	20595360
2	2	Brazil	1999	37737	172006362
3	3	Brazil	2000	80488	174504898
4	4	China	1999	212258	1272815272
5	5	China	2000	213766	1280428583
6	0	Afghanistan	1999	745	19987071
7	1	Afghanistan	2000	2666	20595360
8	2	Brazil	1999	37737	172006362
9	3	Brazil	2000	80488	174504898
10	4	China	1999	212258	1272815272
11	5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

		axis columns		
		0	1	2
axis index	0	0,0	0,1	0,2
	1	1,0	1,1	1,2
	2	2,0	2,1	2,2

Concatenating rows

- A shortcut for the **concat()** function when two dataframes are joined is the **append()** function

```
dataframe.append(dataframe)
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Concatenating columns

- With the **axis** parameter we can concatenate columns instead of rows

```
pd.concat((dataset_1, dataset_2), axis = 'columns')
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

dataset_1			dataset_2		
	Country	Year		Cases	Population
0	Afghanistan	1999	0	745	19987071
1	Afghanistan	2000	1	2666	20595360
2	Brazil	1999	2	37737	172006362
3	Brazil	2000	3	80488	174504898
4	China	1999	4	212258	1272815272
5	China	2000	5	213766	1280428583

		axis columns		
		0	1	2
axis index	0	0,0	0,1	0,2
	1	1,0	1,1	1,2
	2	2,0	2,1	2,2

Concatenating columns - Rankings

```
rank = dataframe.rank().rename(columns = lambda column: "Rank " + column)
rank
```

	Rank Country	Rank Year	Rank Cases	Rank Population
0	1.5	2.0	1.0	1.0
1	1.5	5.0	2.0	2.0
2	3.5	2.0	3.0	3.0
3	3.5	5.0	4.0	4.0
4	5.5	2.0	5.0	5.0
5	5.5	5.0	6.0	6.0

```
pd.concat((dataframe, rank), axis = 'columns')
```

	Country	Year	Cases	Population	Rank Country	Rank Year	Rank Cases	Rank Population
0	Afghanistan	1999	745	19987071	1.5	2.0	1.0	1.0
1	Afghanistan	2000	2666	20595360	1.5	5.0	2.0	2.0
2	Brazil	1999	37737	172006362	3.5	2.0	3.0	3.0
3	Brazil	2000	80488	174504898	3.5	5.0	4.0	4.0
4	China	1999	212258	1272815272	5.5	2.0	5.0	5.0
5	China	2000	213766	1280428583	5.5	5.0	6.0	6.0

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

		axis columns		
		0	1	2
axis index	0	0,0	0,1	0,2
	1	1,0	1,1	1,2
	2	2,0	2,1	2,2

Exercise 17

- Duplicate the content of the dataset "liga"
- Filter the columns "Puesto", "Equipo", "Zona" and "PartidosJugados"
- Show only the top three positions

	Puesto	Equipo	Zona	PartidosJugados
6	2	Barcelona	Champions	28.0
7	2	Barcelona	Champions	28.0
26	1	Real Madrid	Champions	27.0
27	1	Real Madrid	Champions	27.0
30	3	Sevilla	Champions	28.0
31	3	Sevilla	Champions	28.0



Agenda

- Reading / Writing data
- Exploring a DataFrame
- Renaming headings
- Filtering Columns
- Filtering Rows
- Sorting Data
- Changing Data
- Deleting Data
- Grouping Data
- Concatenating Data
- **Joining Data**
- Pivot Tables



Join

- This operation lets us **join** two datasets by a specific column

```
SELECT A.column_PK, B.column  
FROM table1 a A  
INNER JOIN table2 as B ON A.column_PK = B.column_PK
```



Join

- The **merge()** function allows to make a join between two DataFrames

```
df_capitals = pd.DataFrame(  
    {"Country" : ["Afghanistan", "Brazil", "Spain"],  
     "Capital" : ["Kabul", "Brasilia", "Madrid"]},  
    columns = ["Country", "Capital"]  
)  
df_capitals
```

	Country	Capital
0	Afghanistan	Kabul
1	Brazil	Brasilia
2	Spain	Madrid

Join

- The **merge()** function allows to make a join between two DataFrames

```
pd.merge(dataframe, df_capitals, on = "Country")
```

	Country	Year	Cases	Population	Capital
0	Afghanistan	1999	745	19987071	Kabul
1	Afghanistan	2000	2666	20595360	Kabul
2	Brazil	1999	37737	172006362	Brasilia
3	Brazil	2000	80488	174504898	Brasilia

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

	Country	Capital
0	Afghanistan	Kabul
1	Brazil	Brasilia
2	Spain	Madrid



(*) In case of multiple columns it is possible to specify an array of fields

Join

- The parameters **left_on** and **right_on** allow to specify different names in both dataframes

```
pd.merge(dataframe, df_capitals,  
         left_on = "Country",  
         right_on = "Country")
```

	Country	Year	Cases	Population	Capital
0	Afghanistan	1999	745	19987071	Kabul
1	Afghanistan	2000	2666	20595360	Kabul
2	Brazil	1999	37737	172006362	Brasilia
3	Brazil	2000	80488	174504898	Brasilia

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

	Country	Capital
0	Afghanistan	Kabul
1	Brazil	Brasilia
2	Spain	Madrid

Join

- The parameter **how** allows to specify different join methods: 'left', 'right', 'outer', 'inner'

```
pd.merge(dataframe, df_capitals, on = "Country", how = "inner")
```

	Country	Year	Cases	Population	Capital
0	Afghanistan	1999	745	19987071	Kabul
1	Afghanistan	2000	2666	20595360	Kabul
2	Brazil	1999	37737	172006362	Brasilia
3	Brazil	2000	80488	174504898	Brasilia

```
pd.merge(dataframe, df_capitals, on = "Country", how = "outer")
```

	Country	Year	Cases	Population	Capital
0	Afghanistan	1999.0	745.0	1.998707e+07	Kabul
1	Afghanistan	2000.0	2666.0	2.059536e+07	Kabul
2	Brazil	1999.0	37737.0	1.720064e+08	Brasilia
3	Brazil	2000.0	80488.0	1.745049e+08	Brasilia
4	China	1999.0	212258.0	1.272815e+09	NaN
5	China	2000.0	213766.0	1.280429e+09	NaN
6	Spain	NaN	NaN	NaN	Madrid

```
pd.merge(dataframe, df_capitals, on = "Country", how = "left")
```

	Country	Year	Cases	Population	Capital
0	Afghanistan	1999	745	19987071	Kabul
1	Afghanistan	2000	2666	20595360	Kabul
2	Brazil	1999	37737	172006362	Brasilia
3	Brazil	2000	80488	174504898	Brasilia
4	China	1999	212258	1272815272	NaN
5	China	2000	213766	1280428583	NaN

```
pd.merge(dataframe, df_capitals, on = "Country", how = "right")
```

	Country	Year	Cases	Population	Capital
0	Afghanistan	1999.0	745.0	19987071.0	Kabul
1	Afghanistan	2000.0	2666.0	20595360.0	Kabul
2	Brazil	1999.0	37737.0	172006362.0	Brasilia
3	Brazil	2000.0	80488.0	174504898.0	Brasilia
4	Spain	NaN	NaN	NaN	Madrid

Anti Join

- The parameter **indicator** allows to show those rows that are not included in the join

```
pd.merge(dataframe, df_capitals, on = "Country", how = "left", indicator = True)
```

	Country	Year	Cases	Population	Capital	_merge
0	Afghanistan	1999	745	19987071	Kabul	both
1	Afghanistan	2000	2666	20595360	Kabul	both
2	Brazil	1999	37737	172006362	Brasilia	both
3	Brazil	2000	80488	174504898	Brasilia	both
4	China	1999	212258	1272815272	NaN	left_only
5	China	2000	213766	1280428583	NaN	left_only

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

	Country	Capital
0	Afghanistan	Kabul
1	Brazil	Brasilia
2	Spain	Madrid

Anti Join

- The parameter **how** allows to choose in which table you want apply the anti-join

```
pd.merge(dataframe, df_capitals, on = "Country", how = "left",  
         indicator = True) \  
    .query("_merge == 'left_only'") \  
    .drop('_merge', axis='columns')
```

	Country	Year	Cases	Population	Capital
4	China	1999	212258	1272815272	NaN
5	China	2000	213766	1280428583	NaN

```
pd.merge(dataframe, df_capitals, on = "Country", how = "right",  
         indicator = True) \  
    .query("_merge == 'right_only'") \  
    .drop('_merge', axis='columns')
```

	Country	Year	Cases	Population	Capital
4	Spain	NaN	NaN	NaN	Madrid

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

	Country	Capital
0	Afghanistan	Kabul
1	Brazil	Brasilia
2	Spain	Madrid

Exercise 18

- Create a DataFrame called “equipos” with the content of the Excel file “Equipos.xlsx”

	Equipo	Provincia	Comunidad Autónoma
0	Real Madrid	Madrid	Madrid
1	Barcelona	Barcelona	Cataluña
2	Sevilla	Sevilla	Andalucía
3	Atlético Madrid	Madrid	Madrid
4	Villarreal	Castellón	Comunidad Valenciana

- In the DataFrame “liga”, assign new columns with the province of the team and another one with its autonomous community (materialize the columns)

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos	DiferenciaGoles	PorcentajeGoles	PorcentajeVictorias	PuntosAcum	Zona	DiferenciaPuntos	Provincia	Comunidad Autónoma
0	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0	43.0	0.087871	0.740741	65.0	Champions	2.0	Madrid	Madrid
1	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0	56.0	0.100248	0.678571	128.0	Champions	6.0	Barcelona	Cataluña
2	3	Sevilla	28.0	17.0	6.0	5.0	52.0	34.0	57.0	18.0	0.064356	0.607143	185.0	Champions	2.0	Sevilla	Andalucía
3	4	Atlético Madrid	28.0	16.0	7.0	5.0	52.0	23.0	55.0	29.0	0.064356	0.571429	240.0	Champions	7.0	Madrid	Madrid
4	5	Villarreal	28.0	13.0	9.0	6.0	39.0	20.0	48.0	19.0	0.048267	0.464286	288.0	Normal	0.0	Castellón	Comunidad Valenciana

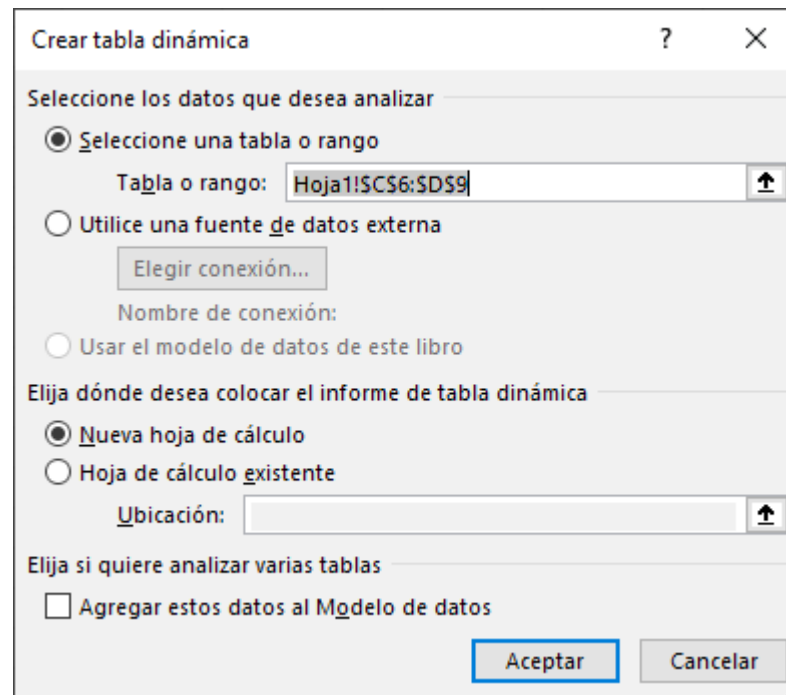
Agenda

- Reading / Writing data
- Exploring a DataFrame
- Renaming headings
- Filtering Columns
- Filtering Rows
- Sorting Data
- Changing Data
- Deleting Data
- Grouping Data
- Concatenating Data
- Joining Data
- **Pivot Tables**



Pivot Tables

- The function **pivot_table** () allows us to create a table where we apply a series of grouping functions in a set of values and categories at the same time.



Pivot Tables

- The simplest pivot table must have a DataFrame and an **index** parameter

```
dataframe.pivot_table(  
    index = "Country"  
)
```

	Cases	Population	Year
Country			
Afghanistan	1705.5	2.029122e+07	1999.5
Brazil	59112.5	1.732556e+08	1999.5
China	213012.0	1.276622e+09	1999.5

```
dataframe.pivot_table(  
    index = ["Year", "Country"]  
)
```

	Cases	Population
Year	Country	
1999	Afghanistan	745 19987071
	Brazil	37737 172006362
	China	212258 1272815272
2000	Afghanistan	2666 20595360
	Brazil	80488 174504898
	China	213766 1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583



Pivot Tables

- You can add the **values** parameter to filter the values that are shown (by default it will show all the numerical values)

```
dataframe.pivot_table(  
    index = "Country"  
)
```

	Cases	Population	Year
Country			
Afghanistan	1705.5	2.029122e+07	1999.5
Brazil	59112.5	1.732556e+08	1999.5
China	213012.0	1.276622e+09	1999.5

```
dataframe.pivot_table(  
    index = "Country",  
    values = "Cases"  
)
```

	Cases
Country	
Afghanistan	1705.5
Brazil	59112.5
China	213012.0

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Pivot Tables

- By default, **pivot_table** uses the **mean** as aggregate function but it can be changed with the **aggfunc** parameter

```
dataframe.pivot_table(  
    index = ["Country"],  
    values = ["Cases"]  
)
```

Cases	
Country	
Afghanistan	1705.5
Brazil	59112.5
China	213012.0

```
dataframe.pivot_table(  
    index = "Country",  
    values = "Cases",  
    aggfunc = "sum"  
)
```

Cases	
Country	
Afghanistan	3411
Brazil	118225
China	426024

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Pivot Tables

- You can use a list of aggregate functions instead a single value

```
dataframe.pivot_table(  
    index = "Country",  
    values = "Cases",  
    aggfunc = [np.sum, "mean"]  
)
```

	sum	mean
	Cases	Cases
Country		
Afghanistan	3411	1705.5
Brazil	118225	59112.5
China	426024	213012.0

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Pivot Tables

- Using the appropriate function you can aggregate non numerical columns ...

```
dataframe.pivot_table(  
    index = "Year",  
    values = "Country",  
    aggfunc = lambda x: ' / '.join(x)  
)
```

	Country
Year	
1999	Afghanistan / Brazil / China
2000	Afghanistan / Brazil / China

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Pivot Tables

- The **columns** parameter provide an additional way to segment the actual values you care about.

```
dataframe.pivot_table(  
    index = "Country",  
    columns = "Year",  
    values = "Cases",  
    aggfunc = np.sum  
)
```

	Year	1999	2000
Country			
Afghanistan		745	2666
Brazil		37737	80488
China		212258	213766

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Exercise 19

- About the DataFrame “liga”,
- Create a pivot table with the average points per autonomous community
- Sort the result from highest to lowest

Puntos	
Comunidad Autónoma	
Cataluña	51.500000
Madrid	48.666667
Pais Vasco	43.250000
Comunidad Valenciana	39.000000
Canarias	35.000000
Andalucia	33.500000
Galicia	32.500000
Asturias	21.000000
Navarra	11.000000



Exercise 20

- Create a pivot table with the average of “Puntos” and “DiferenciaGoles” by autonomous community and province

		DiferenciaGoles	Puntos
Comunidad Autónoma	Provincia		
Andalucía	Granada	-33.000000	19.000000
	Málaga	-12.000000	27.000000
	Sevilla	2.500000	44.000000
Asturias	Asturias	-26.000000	21.000000
Canarias	Las Palmas	-1.000000	35.000000
Cataluña	Barcelona	28.500000	51.500000
Comunidad Valenciana	Castellón	19.000000	48.000000
	Valencia	-13.000000	30.000000
Galicia	A Coruña	-12.000000	27.000000
	Pontevedra	-5.000000	38.000000
Madrid	Madrid	17.666667	48.666667
Navarra	Navarra	-39.000000	11.000000
País Vasco	Guipúzcoa	4.000000	44.500000
	Vizcaya	3.000000	44.000000
	Álava	-4.000000	40.000000

Exercise 21

- Create a pivot table comparing the “Zona” and the autonomous community, where the average points are shown, using a fill value of 0 (fill_value)

Zona	Champions	Normal	Descenso
Comunidad Autónoma			
Andalucía	57	29.00	19
Asturias	0	21.00	0
Canarias	0	35.00	0
Cataluña	63	40.00	0
Comunidad Valenciana	0	39.00	0
Galicia	0	32.50	0
Madrid	60	26.00	0
Navarra	0	0.00	11
Pais Vasco	0	43.25	0

THANKS FOR YOUR ATTENTION

Daniel Villanueva Jiménez
daniel.villanueva@immune.institute
@dvillaj

