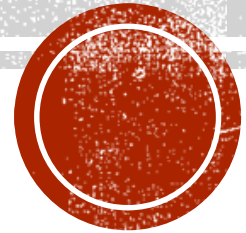
PYTHON - PANDAS

Christian Camilo Urcuqui López, MSc



UNIVERSIDAD ICESI | A OTRO NIVEL

PRESENTACIÓN

Christian Camilo Urcuqui López

Ing. Sistemas, Magister en Informática y Telecomunicaciones

Big Data Professional

Big Data Scientist

Deep Learning Specialization

Grupo de investigación i2t

Líder de investigación y desarrollo

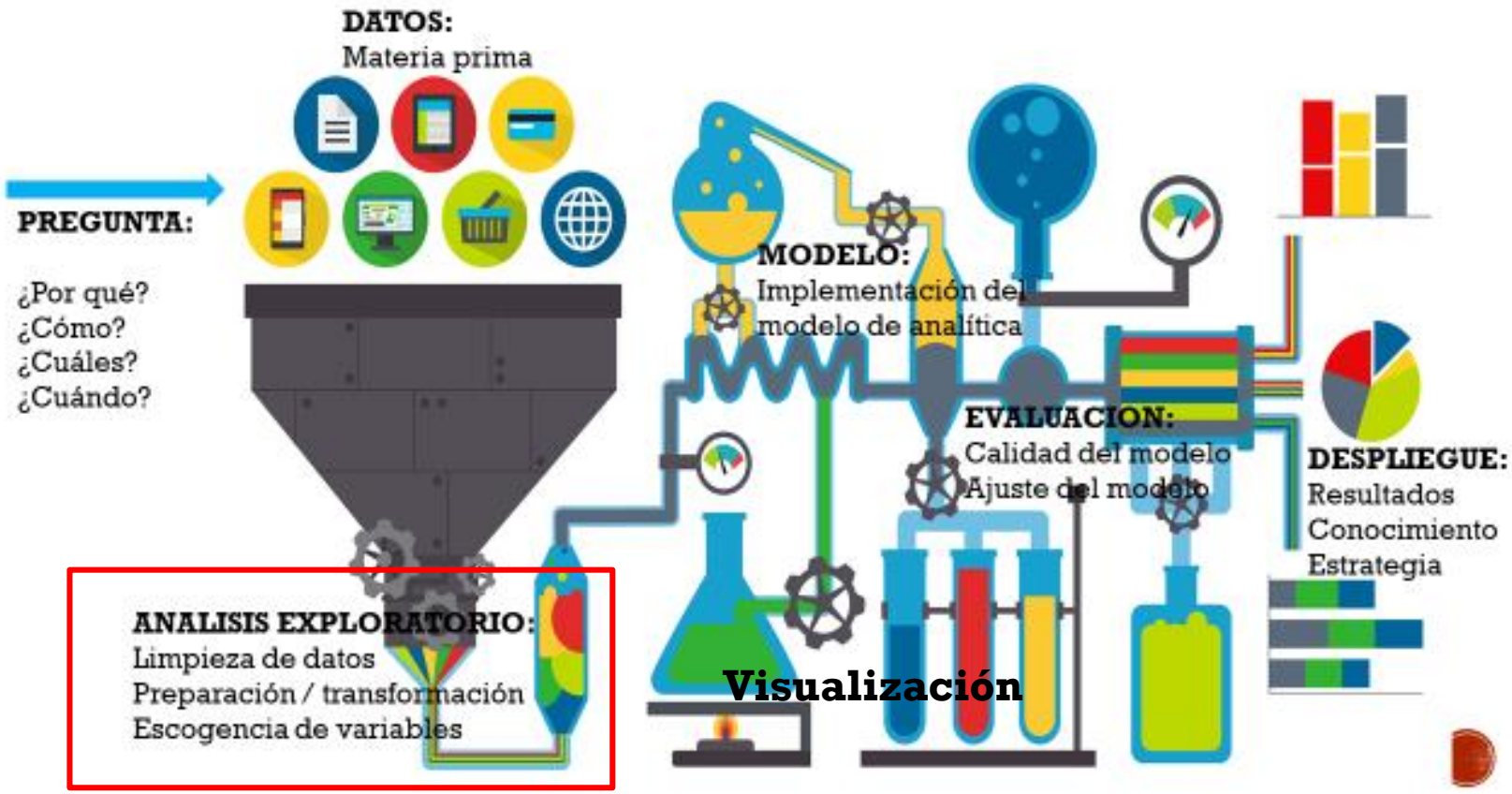
Ciberseguridad y ciencia de datos aplicada

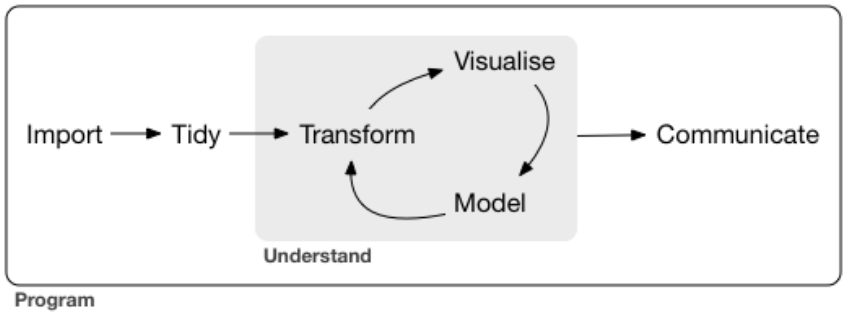
ccurcuqui@icesi.edu.co

COMPETENCIAS

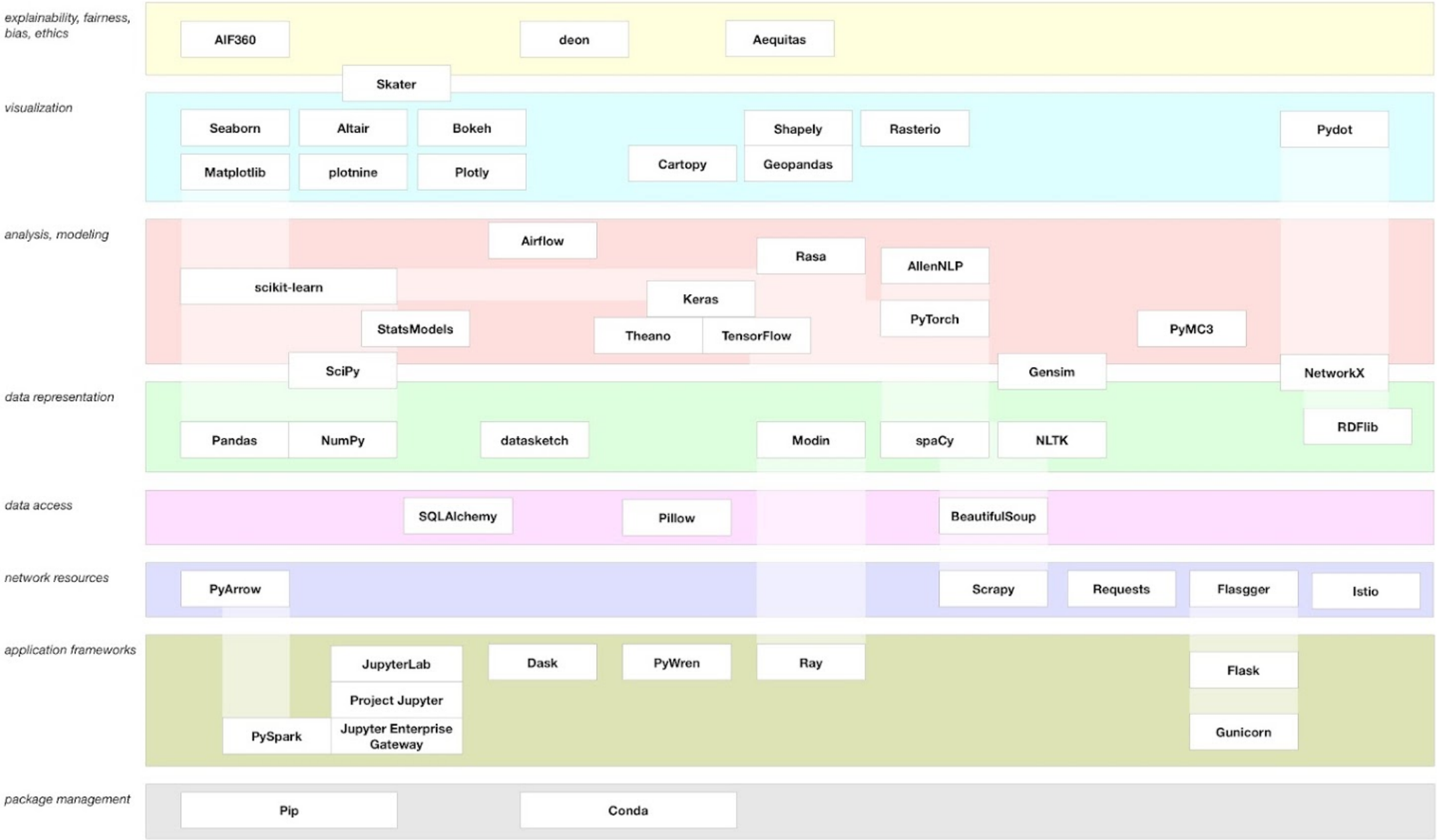
- Utilizar las librerías de Python para proyectos de analítica de datos.
- Numpy
- Pandas
- Matplotlib

CICLO DE VIDA

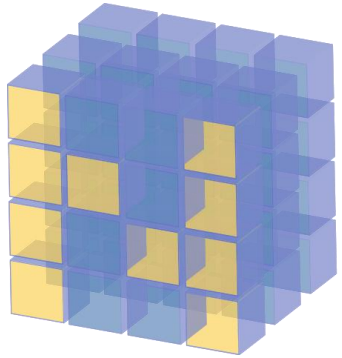




Marco de trabajo típico de un proyecto de ciencia de datos.
R for Data Science


explainability, fairness, bias, ethics
AIF360
deon
Aequitas
Skater
visualization
Seaborn
Altair
Bokeh
Matplotlib
plotnine
Plotly
Cartopy
Shapely
Geopandas
Rasterio
Pydot
analysis, modeling
Airflow
Rasa
AllenNLP
scikit-learn
Keras
PyTorch
StatsModels
Theano
TensorFlow
PyMC3
SciPy
Gensim
NetworkX
data representation
Pandas
NumPy
datasketch
Modin
spaCy
NLTK
RDFlib
data access
SQLAlchemy
Pillow
BeautifulSoup
network resources
PyArrow
Scrapy
Requests
Flasgger
Istio
application frameworks
JupyterLab
Dask
PyWren
Ray
Flask
Project Jupyter
PySpark
Jupyter Enterprise Gateway
Gunicorn
package management
Pip
Conda

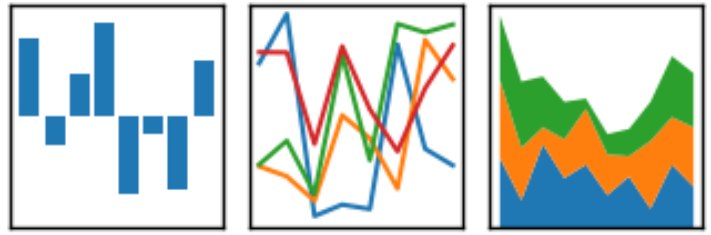
PAQUETES A REVISAR



NumPy

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



matplotlib

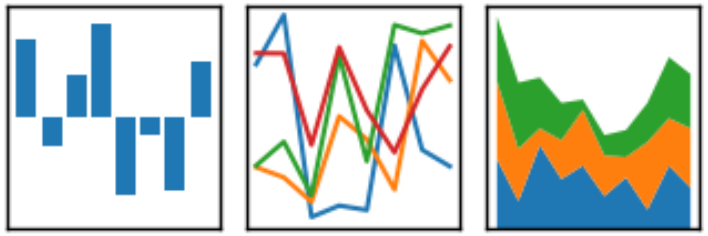
scikits learn

machine learning in Python

SciPy

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



- Pandas es una librería *open source* de estructuras de datos rápidas, flexibles y expresivas para el análisis de información en el lenguaje de programación Python,
- En esta presentación veremos las funciones esenciales para la aplicación de análisis de datos a través de Pandas, además, veremos los primeros pasos de un análisis exploratorio de datos (*Exploratory Data Analysis*) desde la carga de los datos hasta el descubrimiento de elementos de valor en nuestros conjuntos de datos.
- *La gran diferencia entre Pandas y NumPy es que esta librería esta diseñada para la manipulación de datos de distinto tipo en un mismo objeto, es común que ambas herramientas se utilicen juntas ya que Pandas depende en gran medida de la matriz NumPy*.

IMPORTANDO PANDAS

Es común que algunos desarrolladores por estándar usen el seudónimo *pd* al momento de importar el paquete Pandas.

```
import pandas as pd
pd.__version__
```

ESTRUCTURAS DE DATOS

En Pandas podemos encontrar dos tipos de estructuras de datos.

- Series
- DataFrames

SERIES

- Las series representan a un arreglo de secuencia de valores de una sola dimensión que se encuentra asociado a un arreglo de labels (*index*).
- **Un objeto Serie es más flexible que un array NumPy ya que podemos definir nuestras propias etiquetas para los índices, es decir, podemos utilizar tanto números como letras.**

Índice	Datos
0	'A'
1	'B'
2	'C'
3	'D'
4	'E'

SERIES

Escriba y ejecute las siguientes líneas de código en una celda.

```
serie = Series([3, -8, "a"])

serie
```

Observe los datos y los índices.

SERIES

Con la siguiente línea podemos acceder a los valores de la serie.

serie.values

Podemos ver el rango de índices desde donde inicia, cuanto incrementa y donde finaliza

serie.index

SERIES

Podemos cambiar los índices asignándoles un arreglo del tamaño a la cantidad de datos.

```
serie.index = ["a", "b", "c"]
```

Podemos asignar los índices al momento de crear una serie

```
serie = Series([3, -8, "a"], index=["a", "b", "c"])
```

SERIES - RECORRIDOS

Podemos recorrer el objeto Serie y obtener los valores.

Serie[índice]

```
serie = Series([3, -8, "a"], index=["a", "b", "c"])
```

Índice	Datos
‘a’	3
‘b’	-8
‘c’	‘a’

serie[‘a’]

SERIES - RECORRIDOS

Podemos realizar filtros y otras operaciones.

Serie[operación lógica]

```
serie = Series([3, -8, "a"], index=["a", "b", "c"])
```

Índice	Datos
‘a’	3
‘b’	-8
‘c’	‘a’

```
serie[serie == ‘c’]
```

NO PODRIAMOS APLICAR A ESTA SERIE

```
serie[serie >= -10]
```

Pandas

SERIES - RECORRIDOS

Podemos realizar filtros y otras operaciones.

Serie operación aritmética

serie = Series([3, -8, 2])

Índice	Datos
0	3
1	-8
2	2

Ejecute las siguientes operaciones
serie * 2
serie + serie
serie ** 2

SERIES

Otra forma de crear una Serie es a través de un diccionario.

```
sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah':5000}
series = Series(sdata)
series
```

Los índices serán cada una de las llaves, del mismo modo que los valores del directorio estarán asociados ahora a los datos de la Serie.

Índice	Datos
'Ohio'	35000
'Texas':	71000
'Oregon'	16000
'Utah'	5000

SERIES

Ahora, cree una nueva serie con el diccionario de sdata y el siguiente vector de índices.

```
sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah':5000}
states = ['California', 'Ohio', 'Oregon', 'Texas']
```

Observe las diferencias

OBSERVACIÓN

Debido a que ingresamos una llave nueva (california) al que no tiene un valor asignado en sdata obtenemos como resultado un NaN (valor faltante) y el valor de la llave 'Utah' se pierde.

SERIES

Realice la siguiente operación

```
series2 = Series(sdata)
print(series)
print()
print(series2)
print()
print(series + series2)
```

OBSERVACIÓN

Del anterior ejemplo podemos ver que las operaciones de registros con NaN implica como resultado NaN

DATAFRAME

- Un DataFrame es una estructura similar a una tabla de datos rectangular donde cada columna puede tener distintos tipos de datos (numéricos, caracteres, booleanos).
- Un DataFrame tiene tanto una indexación para sus filas y columnas por lo cual permite almacenar en memoria más de dos bloques dimensionales a diferencia de otros tipos de estructuras.

Pandas

DATAFRAME - CREAR

Observemos distintas formas para crear un DataFrame.

```
state = Series( ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'], name="state")

year = Series([2000, 2001, 2002, 2001, 2002, 2003], name="year")

pop =  Series([1.5, 1.7, 3.6, 2.4, 2.9, 3.2], name="pop")
```

DATAFRAME - CREAR

Observemos distintas formas para crear un DataFrame.

```
data = pd.concat([state, year, pop], axis=1)
```

El axis=1 define que la concatenación se va a realizar por columnas.

DATAFRAME - CREAR

Serie Serie Serie

state	year	pop
2000	Ohio	1.5
2001	Ohio	1.7
2002	Ohio	3.6
2001	Nevada	2.4
2002	Nevada	2.9
2003	Nevada	3.2

DATAFRAME - CREAR

Podemos leer un archivo (CSV) y cargarlo como un DataFrame

```
parks =  pd.read_csv("parks.csv")
```

Para obtener una vista de los primeros cinco registros

```
parks.head()
```

Para obtener una vista de los últimos cinco registros

```
parks.tail()
```

DATAFRAME HEAD

Columns

index

	Park Code	Park Name	State	Acres	Latitude	Longitude
0	ACAD	Acadia National Park	ME	47390	44.35	-68.21
1	ARCH	Arches National Park	UT	76519	38.68	-109.57
2	BADL	Badlands National Park	SD	242756	43.75	-102.50
3	BIBE	Big Bend National Park	TX	801163	29.25	-103.25
4	BISC	Biscayne National Park	FL	172924	25.65	-80.08

Pandas

DATAFRAME

- Por favor creemos un nuevo DataFrame y asígnele el nombre a la variable *df*

df

state	year	pop
Ohio	2000	1.5
Ohio	2001	1.7
Ohio	2002	3.6
Nevada	2001	2.4
Nevada	2002	2.9
Nevada	2003	3.2

DATAFRAME – CAMBIANDO NOMBRES

Podemos cambiar los nombres de las columnas

1. Método 1

```
# cambiemos los nombres de las columnas a español
df.columns = ["Estado", "Año", "Población“]
```

2. Método 2

```
df = df.rename({'state':'Estado', 'year':'Año', 'pop':'Población'},
axis='columns')
```

Debe tener todas las columnas

No es necesario escribir todas columnas

DATAFRAME – CAMBIANDO NOMBRES

Cambiemos los nombres de los índices de las filas

df.index = ["Registro0", "Registro1", "Registro2", "Registro3", "Registro4", "Registro5"]

	state	year	pop
Registro0	Ohio	2000	1.5
Registro1	Ohio	2001	1.7
Registro2	Ohio	2002	3.6
Registro3	Nevada	2001	2.4
Registro4	Nevada	2002	2.9
Registro5	Nevada	2003	3.2

La cantidad de índices debe ser si o si a la misma cantidad de registros

DATAFRAME – CARGA Y ESCRITURA DE DATOS

Podemos tener distintas situaciones donde debemos cargar conjuntos de datos (posiblemente de distinto tipo) a nuestra computadora local, observemos algunos ejemplos de Pandas para cargar datos.

DATAFRAME – CARGA Y ESCRITURA DE DATOS

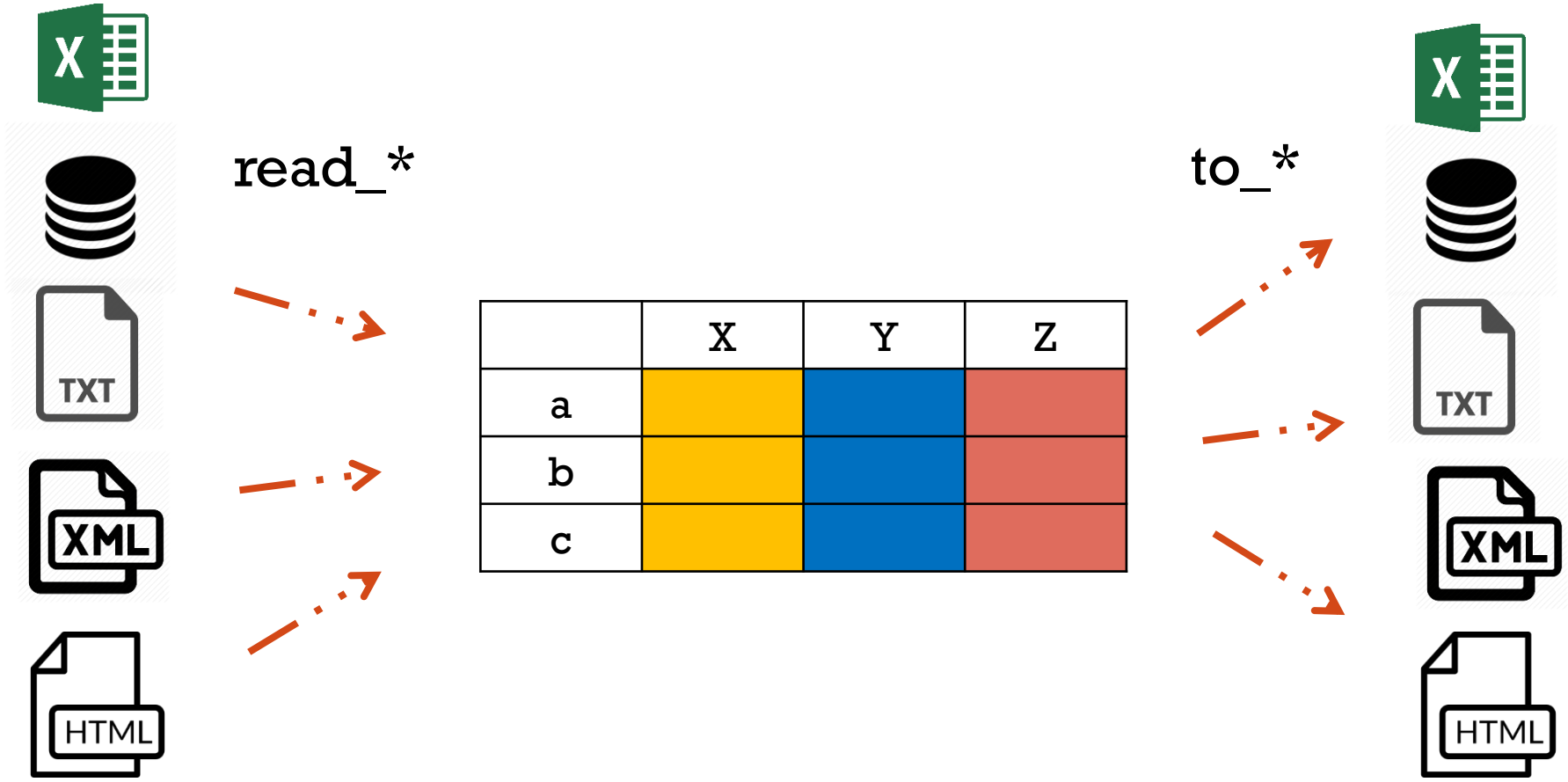
Pandas tiene métodos dedicados a procesar distintas fuentes de datos, esta y más información la podemos encontrar en su página web:

https://pandas.pydata.org/pandas-docs/stable/reference/io.html

Algunos de los tipos de archivos que podemos tratar son los siguientes:

- HTML
- JSON
- Excel
- SQL
- SAS
- Google BigQuery
- STATA

DATAFRAME – CARGA Y ESCRITURA DE DATOS



Pandas

DATAFRAME – CARGA Y ESCRITURA DE DATOS

No se olvide de revisar siempre la ayuda

?pd.read_csv

help(pd.read_csv)

Hay un conjunto de parámetros que le pueden ser de utilidad, pero el uso depende de como este la fuente de los datos y que deseamos realizar con ellos.

DATAFRAME – CARGA Y ESCRITURA DE DATOS

Si tenemos disponible un URL (enlace) donde podemos descargar y cargar los datos en memoria, Pandas nos ofrece un método para realizar estas operaciones. Existen múltiples repositorios abiertos a su uso para el trabajo de ciencia de datos, entre estos podemos citar:

- Kaggle
- University of California Machine Learning (UCI Machine Learning)
- IEEE DataPort
- World Bank Dataset
- AWS datasets
- Fuentes abiertas de instituciones públicas (por ejemplo, data.gov.in y datos.gov.co)

DATAFRAME – CARGA Y ESCRITURA DE DATOS

Cargamos un conjunto de datos (tipo .data) y les asignamos los nombres a cada columna

```
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data', header=None, names=["sepal_length","sepal_width", "petal_length","petal_width","class"])

df.head(5)
```

OBSERVACIÓN

Le recomendamos siempre revisar la fuente de los datos antes de cargarla a un DataFrame

DATAFRAME – CARGA Y ESCRITURA DE DATOS

Cargamos un CSV desde un URL, preste atención que ahora definimos el separador (por defecto procesa datos separados por comas)

```
pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv', sep=';')
```

Pandas

DATAFRAME – CARGA Y ESCRITURA DE DATOS

Para el siguiente ejemplo cargaremos un Excel que tiene dos sheets (hojas), debemos tener presente que por defecto trae la posición 0.

```
df = pd.read_excel("../../../datasets/example_sheets2.xlsx")
```

DATAFRAME – CARGA Y ESCRITURA DE DATOS

Cargamos el segundo sheet desde el índice

```
df = pd.read_excel("../../../datasets/example_sheets2.xlsx", sheet_name=1)
```

o el identificador.

```
df = pd.read_excel("../../../datasets/example_sheets2.xlsx",
                   sheet_name="Session2")
```

Pandas

DATAFRAME – CARGA Y ESCRITURA DE DATOS

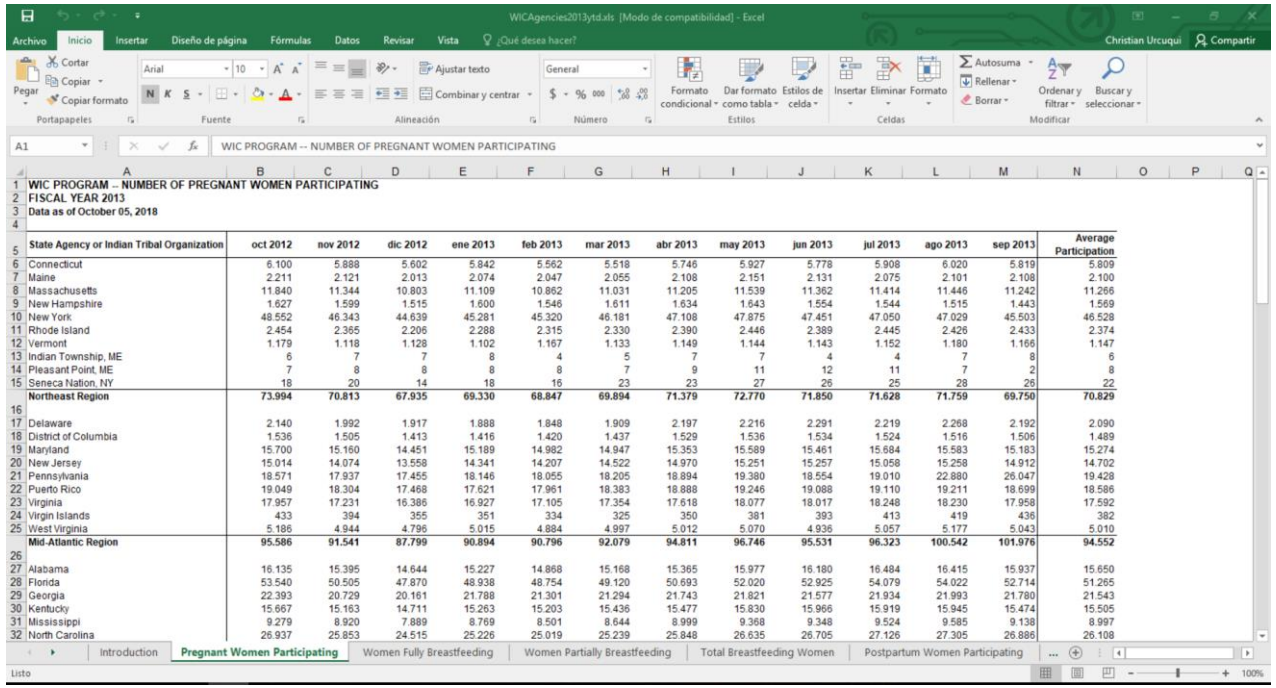
Ahora veamos la diferencia de cargar los datos con el sheet igual a **None**

```
df = pd.read_excel("../../../datasets/example_sheets2.xlsx", sheet_name=None)
type(df)
```

Obtenemos un diccionario donde cada key corresponde a un identificador sheet df.keys()

DATAFRAME – CARGA Y ESCRITURA DE DATOS

Un problema común que podríamos enfrentar es el formato en el archivo Excel, analicemos el siguiente archivo descargado desde el siguiente repositorio:



WIC PROGRAM -- NUMBER OF PREGNANT WOMEN PARTICIPATING
FISCAL YEAR 2013
Data as of October 05, 2018

State Agency or Indian Tribal Organization	oct 2012	nov 2012	dic 2012	ene 2013	feb 2013	mar 2013	abr 2013	may 2013	jun 2013	jul 2013	ago 2013	sep 2013	Average Participation
Connecticut	6.100	5.888	5.602	5.842	5.562	5.518	5.746	5.927	5.778	5.908	6.020	5.819	5.809
Maine	2.211	2.121	2.013	2.074	2.047	2.055	2.108	2.151	2.131	2.075	2.101	2.108	2.100
Massachusetts	11.840	11.344	10.803	11.109	10.862	11.031	11.205	11.539	11.362	11.414	11.446	11.242	11.266
New Hampshire	1.627	1.599	1.515	1.600	1.546	1.611	1.634	1.643	1.554	1.544	1.515	1.443	1.569
New York	48.552	46.343	44.639	45.281	45.320	46.181	47.108	47.875	47.451	47.050	47.029	45.503	46.528
Rhode Island	2.454	2.365	2.206	2.288	2.315	2.330	2.390	2.446	2.389	2.445	2.426	2.433	2.374
Vermont	1.179	1.118	1.128	1.102	1.167	1.133	1.149	1.144	1.143	1.152	1.180	1.166	1.147
Indian Township, ME	6	7	7	8	4	5	7	7	4	4	7	8	6
Pleasant Point, ME	7	8	8	8	8	7	9	11	12	11	7	2	8
Seneca Nation, NY	18	20	14	18	16	23	23	27	26	25	28	26	22
Northeast Region	**73.994**	**70.813**	**67.935**	**69.330**	**68.847**	**69.894**	**71.379**	**72.770**	**71.850**	**71.628**	**71.759**	**69.750**	**70.829**
Delaware	2.140	1.992	1.917	1.888	1.848	1.909	2.197	2.216	2.291	2.219	2.268	2.192	2.090
District of Columbia	1.536	1.505	1.413	1.416	1.420	1.437	1.529	1.536	1.534	1.524	1.516	1.506	1.489
Maryland	15.700	15.160	14.451	15.189	14.982	14.947	15.353	15.589	15.461	15.684	15.583	15.183	15.274
New Jersey	15.014	14.074	13.558	14.341	14.207	14.522	14.970	15.251	15.257	15.058	15.258	14.912	14.702
Pennsylvania	18.571	17.937	17.455	18.146	18.055	18.205	18.894	19.380	18.554	19.010	22.880	26.047	19.428
Puerto Rico	19.049	18.304	17.468	17.621	17.961	18.383	18.888	19.246	19.088	19.110	19.211	18.699	18.586
Virginia	17.957	17.231	16.386	16.927	17.105	17.354	17.618	18.077	18.017	18.248	18.230	17.958	17.592
Virgin Islands	433	394	355	351	334	325	350	381	393	413	419	436	382
West Virginia	5.186	4.944	4.796	5.015	4.884	4.997	5.012	5.070	4.936	5.057	5.177	5.043	5.010
Mid-Atlantic Region	**95.586**	**91.541**	**87.799**	**90.894**	**90.796**	**92.079**	**94.811**	**96.746**	**95.531**	**96.323**	**100.542**	**101.976**	**94.552**
Alabama	16.135	15.395	14.644	15.227	14.868	15.168	15.365	15.977	16.180	16.484	16.415	15.937	15.650
Florida	53.540	50.505	47.870	48.938	48.754	49.120	50.693	52.020	52.925	54.079	54.022	52.714	51.265
Georgia	22.393	20.729	20.161	21.788	21.301	21.294	21.743	21.821	21.577	21.934	21.993	21.780	21.543
Kentucky	15.667	15.163	14.711	15.263	15.203	15.436	15.477	15.830	15.966	15.919	15.945	15.474	15.505
Mississippi	9.279	8.920	7.889	8.769	8.501	8.644	8.999	9.368	9.348	9.524	9.585	9.138	8.997
North Carolina	26.937	25.853	24.515	25.226	25.019	25.239	25.848	26.635	26.705	27.126	27.305	26.886	26.108

WICAgencies2013ytd.xls

https://www.kaggle.com/plsms21/xls-files-all/downloads/xls-files-all.zip

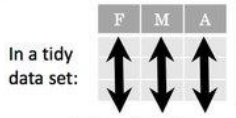
DATAFRAME – CARGA Y ESCRITURA DE DATOS

El archivo tiene muchas hojas; como vimos podemos hacer un DataFrame llamando a una de ellas, pero ¿qué va a pasar con su formato?

```
pd.read_excel("../../../Datasets/xls-files-all/WICAgencies2013ytd.xls", sheet_name='Total Women').head()
```

Data Wrangling with pandas Cheat Sheet http://pandas.pydata.org

Tidy Data – A foundation for wrangling in pandas



Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

M * A

Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```
Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```
Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

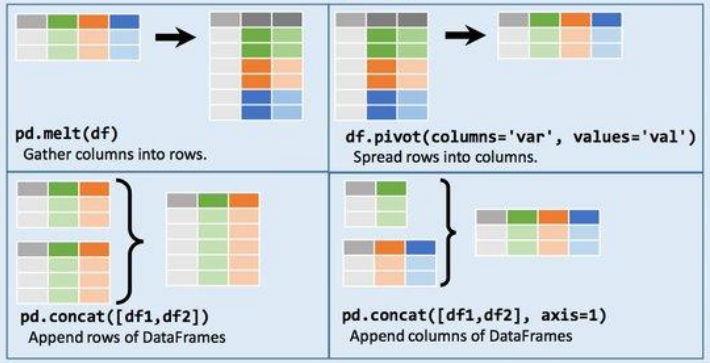
```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
          names=['n','v'])))
```
Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
    )
```

Reshaping Data – Change the layout of a data set



pd.melt(df)
Gather columns into rows.

df.pivot(columns='var', values='val')
Spread rows into columns.

pd.concat([df1,df2])
Append rows of DataFrames

pd.concat([df1,df2], axis=1)
Append columns of DataFrames

df=df.sort_values('mpg')
Order rows by values of a column (low to high).

df=df.sort_values('mpg',ascending=False)
Order rows by values of a column (high to low).

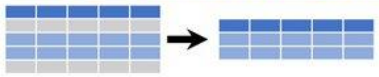
df=df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df=df.sort_index()
Sort the index of a DataFrame

df=df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df=df.drop(['Length','Height'], axis=1)
Drop columns from DataFrame

Subset Observations (Rows)



df[df.Length > 7]
Extract rows that meet logical criteria.

df.drop_duplicates()
Remove duplicate rows (only considers columns).

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

df.sample(frac=0.5)
Randomly select fraction of rows.

df.sample(n=10)
Randomly select n rows.

df.iloc[10:20]
Select rows by position.

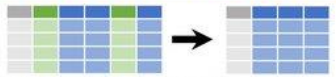
df.nlargest(n, 'value')
Select and order top n entries.

df.nsmallest(n, 'value')
Select and order bottom n entries.

Logic in Python (and pandas)

<	Less than	!=	Not equal to
>	Greater than	df.column.isin(*values*)	Group membership
==	Equals	pd.isnull(*obj*)	Is NaN
<=	Less than or equals	pd.notnull(*obj*)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

Subset Variables (Columns)



df[['width','length','species']]
Select multiple columns with specific names.

df['width'] *or* **df.width**
Select single column with specific name.

df.filter(regex='*regex*')
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples

'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
''^(?!Species$).*'	Matches strings except the string 'Species'

df.loc[:,'x2':'x4']
Select all columns between x2 and x4 (inclusive).

df.iloc[:,[1,2,5]]
Select columns in positions 1, 2 and 5 (first column is 0).

df.loc[df['a'] > 10, ['a','c']]
Select rows meeting logical condition, and only the specific columns .

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

BIBLIOGRAFÍA

- Lutz, M. (2013). *Learning Python: Powerful Object-Oriented Programming*." O'Reilly Media, Inc.".