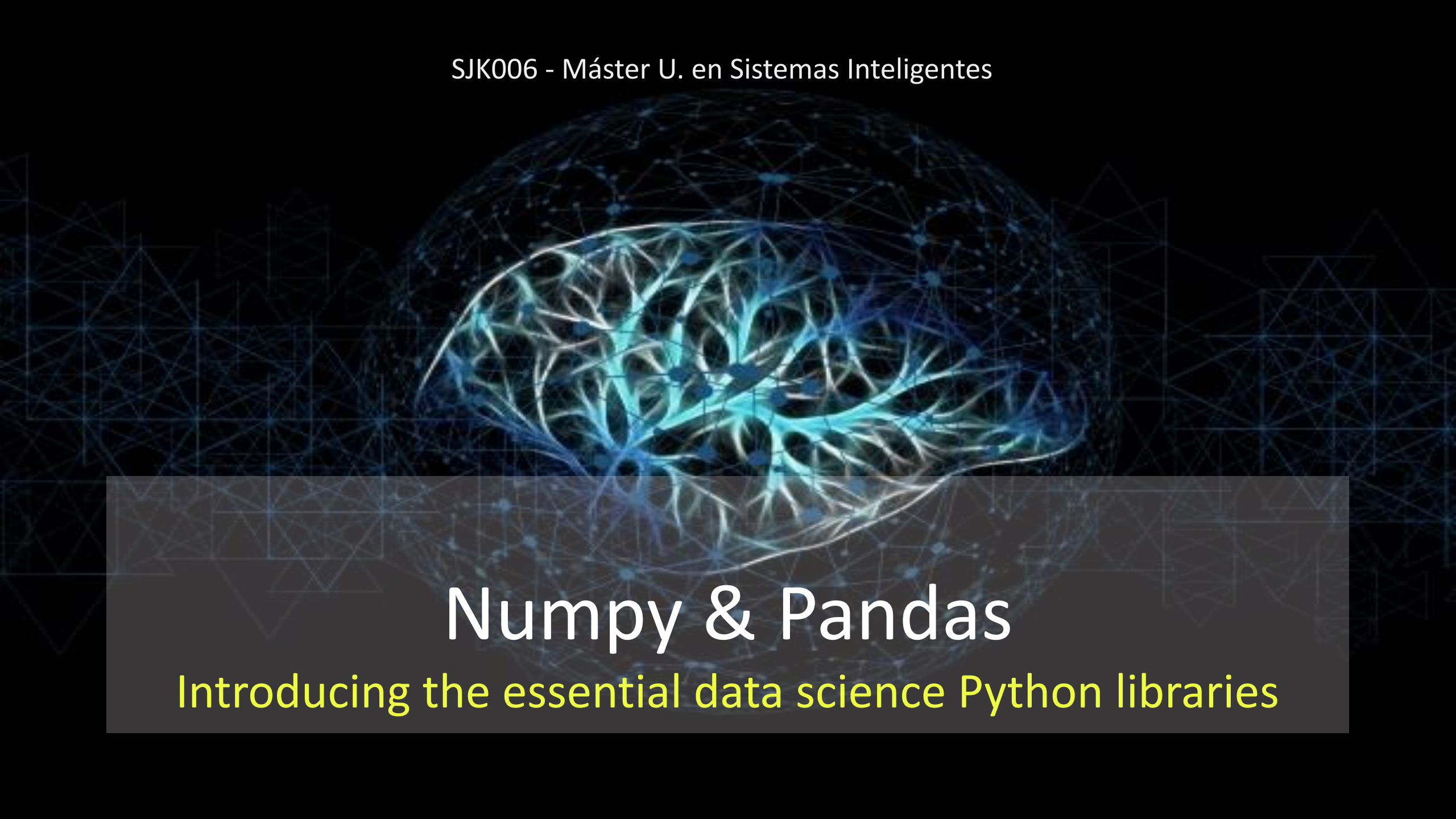
SJK006 - Máster U. en Sistemas Inteligentes



Numpy & Pandas

Introducing the essential data science Python libraries

Numpy

The basis of all data science libraries in Python

You must read:

https://www.nature.com/articles/s41586-020-2649-2

This library gives support to:

- Pandas
- Scikit-learn
- TensorFlow & Torch
- SciPy
- and much more …

Numpy main concepts

- Numpy comprises the classes, constants and functions to manipulate **contiguous numeric arrays**. Usually imported as:

```
import numpy as np
```

- The main class is **array**, which takes a list as input (1-D, 2-D, etc.) and returns an object that contains the contiguous array along with all its basic operations.

```
v = np.array([1.0, 2.0, 3.0])
```

- Basic math operators (+, -, *, /) are overridden:

```
v2 = v + np.ones((1, 3))
```

This is called the "shape" of the array

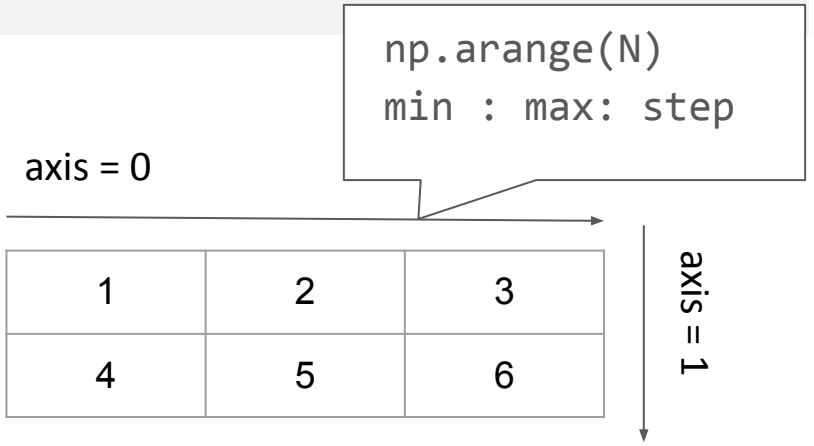
Other array builders

```
np.ones(shape)
np.zeros(shape)
np.full(shape, val)
np.eye(N)
```

Numpy main concepts

- **Slicing** is one of the main operations over arrays:

```
A = np.array([[1, 2, 3], [4, 5, 6]])
A[0, 2]
A[0, : ]
A[ : , 0 : 2]
A[ : , :, np.newaxis]            #adds a new dimension (axis)
A[0, 0 : 2] = np.array([3, 2]) #assign a slice
```



- **Broadcasting** is a powerful mechanism to operate over arrays of different shapes:

```
v = np.ones((3, 3)) * 5

A = np.array([[1, 2], [3, 4]])
B = np.array([10, 20])
result = A + B
```

Broadcasting perform *element-wise* operations without the need of reshaping

Numpy main functions

- **Constants:** `np.nan  np.Infinity  np.pi  np.e`
- **Aggregations** can be done in any of the array axes:

```
A = np.array([[1, 2, 3], [4, 5, 6]])

np.sum(A, axis=0) ## equivalent to A.sum(axis=0)

Aggregators: np.min,  np.max, np.mean, np.median, np.argmin, np.argmax
```

- **Statistics**: `np.prod, np.std,  np.var,  np.corrcoef`
  `np.cov,  np.percentile,  np.histogram`
- Other interesting operations:

`np.dot = @` (dot product) `A.T` (transpose) np.isnan

Reshaping & sorting

Reshaping in NumPy involves changing the shape or dimensions of an existing array while keeping the total number of elements constant.

`reshape(shape)`	rearranges elements	`A.reshape((3, 2))`
`np.resize(A,shape)`	repeat sequence till complete new shape	`np.resize(A, (3, 3))`
`flatten(order=.)`	remove dimensions following a given order	`A.flatten()`
`ravel(order=.)`	same as flatten but creating a view instead of a copy	`B = np.ravel(A)` `B[5] = 3 #modifies A`
`sort(A, axis=.)`	copy with sorted elements	`np.sort(A, axis=0)`
`argsort(A, axis=.)`	copy with the indexes of sorted elements	`np.argsort(A)`

Your can test these functions with A = np.arange(6)

Numpy and linear algebra

Linear algebra is the basis of many machine learning methods.

numpy.linalg contains the main functions to perform basic linear algebra operations:

Determinant of a matrix, Inverse of a matrix, Product of two matrices, Eigenvalues, Normalization, Solving equations, etc.

- https://numpy.org/doc/stable/reference/routines.linalg.html

Pandas

A superset of Numpy to deal with tidy data

https://pandas.pydata.org/docs/reference/index.html

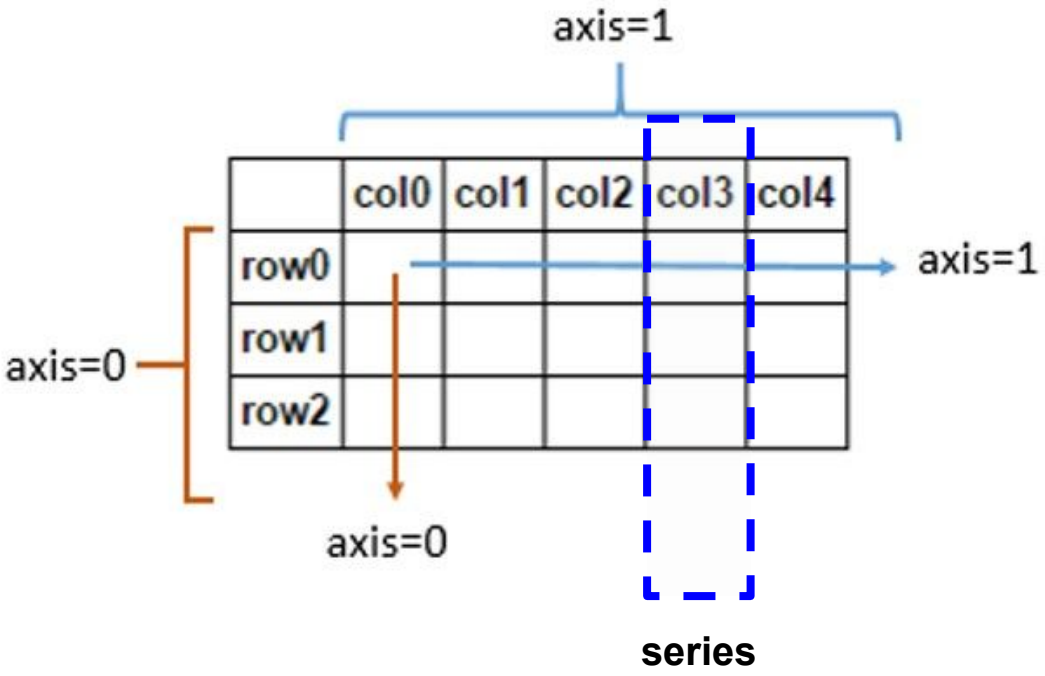
Pandas main classes

- ***Module Pandas***: comprises all the elements to deal with ***tidy data***

- ***Series***: building block consisting of a data series ordered by an **index**. The index can be either numerical or categorical (strings).

- ***DataFrame***: ordered set of named series sharing the same index.

Dataframe structure



Tidy data:
Rows represent observations and columns represent the variables of the analysis/prediction.

DataFrames allow complex orderings of data to perform analytical tasks like grouping, pivoting, multi-rows and multi-columns.

Following numpy notation, **axis 0** represents columns, which are in turn **series**.

Rows (**axis 1**) are identified by a shared index.

Importing data

Pandas provides a great variety of methods for importing data into dataframes:

```
import pandas as pd

df = pd.read_csv(csv-file or web-URL, usecols=...)
df = pd.read_csv(tsv-file, sep='\t')

df = pd.read_excel(file.xlsx, …)
df = pd.read_html(web-file, …)
df = pd.read_sql(...)
```

First exploration of imported data

Several dataframe methods allows a first exploration of the data:

```
df.info()       #tipos de datos de las columnas y tamaño

df.describe()   #solo columnas numéricas

df.head(n)      #muestra las (n) primeras

df.tail(n)      #muestra las (n) últimas filas
```

Missing data

Null values are reported with the constant NaN.

Some functions are provided to detect and impute null values.

```
df.isna(...)
```

```
df.dropna(axis= ..., inplace=True)
```

```
df.fillna(...)
```

Accessing elements

Pandas use a mix of notations for accessing dataframe elements, namely: dictionary-like and numpy-like access methods.

```
df['col-name'].value_counts()
df['col-name'].unique()

df.loc[index].at[column]    #row-column index

df.loc[ : , [columns]]      #range of indexes and columns
```

Data selection (subsets)

column names	`df['col']` ó `df.col`
equal	`df.age == 10`
conjunction	`(df.age > 10) & (df.age < 30)`
disjunction	`(df.age < 10) | (df.age > 30)`
negation	`~(df.age == 10)`
set	`df.status.isin([0, 3])`


```
df.loc[expression, [cols]]      # subsets of columns
df.loc[expression, [cols]] = ...  # assign with slice
```

Changing indexes

You can perform changes in the index with the following dataframe methods:

```
df.index                    # the series of the index

df.reset_index()            # reset the current index

df.set_index([cols])        # set a new index from existing cols
```

You can also do a lot of things with the indexes (index objects)

Summarising and pivoting data

Two basic methods allow you to group and pivot data:

```
df.groupby(by=[cols])          # group keys become the index

df.pivot_table(...)            # pivot keys become the index
```

Merging data frames

Another usual operation over data frames is the fusion of two or more data frames into a new data frame. The basic operations for doing this are:

`df.join(other, on=., how=.)`	Fastest way to join two dataframes
`df.merge(right, on=., how=.)`	This is the true JOIN defined in SQL-like databases (with all combinations for inner, outer, etc.)
`pd.concat([dataframes], sort=.,join=.)`	The most general way to combine (column or row-wise) several data frames into one