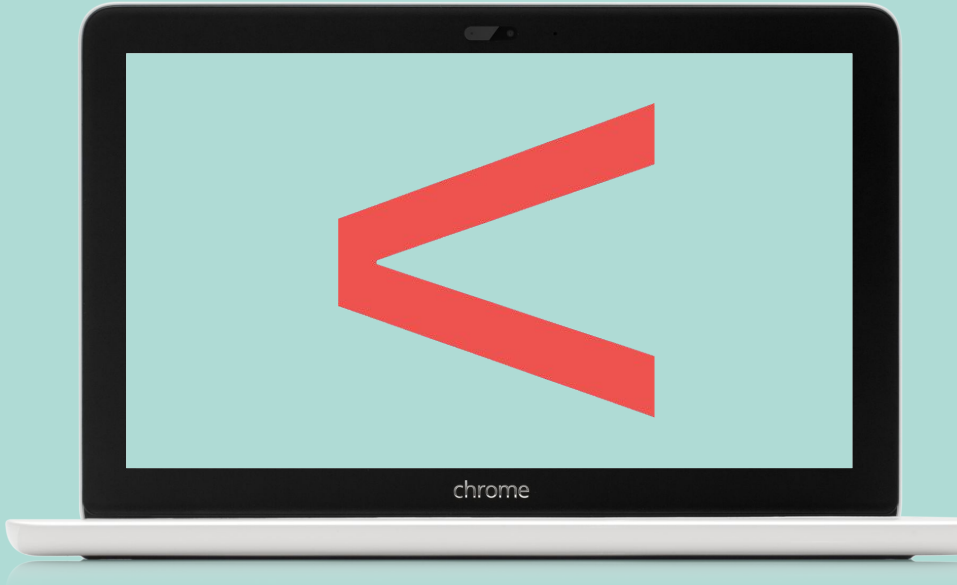
Python libraries: Pandas



What is pandas?

- Pandas is a powerful Python data analysis toolkit.
- An open-source library providing high-performance, easy-to-use data structures.
- It provides **data structures and data analysis tools** designed to make working with “**relational**” **or** “**tabular**” **data** called **Series** (1 dimensional) and **Dataframe** (2 dimensional)

NumPy vs Pandas: Data Structure

NumPy	Pandas
Primarily uses n-dimensional array (ndarray) for data storage and manipulation. This is suitable for homogeneous numerical data types, like integers or floating point values.	Uses Series and DataFrame. Best for tabular data with columns of different types. This is suitable for tabular data, similar to what you'd see in an Excel spreadsheet.

NumPy vs Pandas: Data Type Flexibility

NumPy	Pandas
NumPy arrays are more efficient but they lack flexibility because the entire array must have the same data type.	Pandas' DataFrame and Series can hold multiple data types which provides more flexibility.

NumPy vs Pandas: Handling Missing Values

NumPy	Pandas
NumPy does not have a built-in mechanism to handle missing data.	Pandas provides robust tools for handling missing data, such as isna(), notna(), dropna(), fillna(), etc.

NumPy vs Pandas: Data Analysis Capabilities

NumPy	Pandas
NumPy provides fundamental packages for scientific computing, including basic linear algebra functions, Fourier transforms, advanced random number capabilities. However, it lacks high-level data manipulation tools.	Pandas, built on top of NumPy, offers rich and high-level data manipulation tools, including merging and joining datasets, handling missing data, and filtering data.

NumPy vs Pandas: Size of Data

NumPy	Pandas
For smaller datasets with homogeneous numerical data, NumPy could be faster and uses less memory.	For larger datasets or datasets with heterogeneous data, Pandas provides a more robust and high-performance environment.

NumPy vs Pandas: Integration with Other Libraries

NumPy	Pandas
NumPy arrays are used as the basic data structure by many other scientific or mathematical Python libraries.	Pandas is well integrated with other libraries like Matplotlib, Seaborn for data visualization and Scikit-learn for machine learning, allowing the DataFrame and Series data structures to be passed directly to methods of these libraries.

Pandas vs NumPy

Table of Difference Between Pandas VS NumPy

	PANDAS	NUMPY
1	When we have to work on **Tabular data**, we prefer the *pandas* module.	When we have to work on **Numerical data**, we prefer the *numpy* module.
2	The powerful tools of pandas are **Data frame and Series.**	Whereas the powerful tool of *numpy* is **Arrays.**
3	*Pandas* consume **more memory**.	*Numpy* is **memory efficient.**
4	*Pandas* has a better performance when number of rows is **500K or more.**	*Numpy* has a better performance when number of rows is **50K or less.**
5	Indexing of the *pandas* series is **very slow** as compared to *numpy* arrays.	Indexing of *numpy* Arrays is **very fast**.
6	*Pandas* offers 2d table object called **DataFrame.**	*Numpy* is capable of providing **multi-dimensional arrays.**

For information more about the difference between Numpy and pandas, visit this article.

Source: https://www.geeksforgeeks.org/

Sources:

Pandas:

- Official Pandas Documentation
- Pandas User Guide
- 10 minutes to pandas

NumPy:

- Official NumPy Documentation
- NumPy User Guide
- NumPy: the absolute basics for beginners

Comparisons and usage:

- DataCamp's comparison of NumPy and Pandas
- GeeksforGeeks article comparing Pandas and NumPy

Pandas: main (basic) uses

🧹 Data cleaning and pre-processing

🧬 Data manipulation

⚗️ Applying functions

⏳ Time series analysis

🧮 Performing mathematical operations: statistics, arithmetics, comparison and aggregate functions

📊 Visualizing

🔮 Among other things

pandas' data structures

Data structure refers to the **organization**, **management**, **and storage format of data**, enabling its efficient access and modification.

In other words, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data. For example, an array.

The two most widely used pandas data structures are:
the **Series** and the **DataFrame**

pandas' Series

Series is a one-dimensional **labeled** array capable of holding **different data types**. The axis labels are collectively referred to as the index.

labeled index

```
name          Grumpy Cat
species              cat
YOB                 2012
dtype: object
```

Source: https://pandas.pydata.org/

pandas' Series syntax

```
# Import pandas library
import pandas as pd

# Creating the Series from a list
cat1 = ['Grumpy Cat', 'cat', 2012]
ser = pd.Series(cat1)

# Create the index
cat_index = ['name', 'species',
'YOB']

# Set the index
ser.index = cat_index
```

notice that we're not using the keyword index

Source: https://pandas.pydata.org/

pandas' DataFrame

A DataFrame is a 2-dimensional **labeled data** structure with columns of potentially **different types**. You can think of it like a spreadsheet.

	name	species	YOB
0	Grumpy Cat	cat	2004
1	Garfi the Angry Cat	cat	2012
2	Colonel Meow	cat	2011

index

labels are all the data points or values

feature 'species'

Source: https://pandas.pydata.org/

pandas' DataFrame syntax

```
# Creating a DataFrame from a list
name = [ 'Grumpy Cat', 'Garfi the Angry Cat',
         'Colonel Meow']
species = ['cat','cat', 'cat']
YOB = [2004, 2012, 2011]

# Creating the Series from a dictionary
cat_dict= {  'name':name,
             'species':species,
             'YOB': YOB}
df = pd.DataFrame(cat_dict)
df
```

There are more ways to create a DataFrame. You can explore them in the documentation.

Slicing a pandas' DataFrame - columns

Single column

```
df["name"]
```

```
0              Grumpy Cat
1    Garfi the Angry Cat
2            Colonel Meow
Name: name, dtype: object
```

Each single column in a DataFrame is a Series.

Several columns

```
df[["name", "YOB"]]
```

	name	YOB
0	Grumpy Cat	2004
1	Garfi the Angry Cat	2012
2	Colonel Meow	2011

Slicing a pandas' DataFrame - rows

But, when given a slice, the DataFrame indexing operator selects rows and can do so by integer location or by index label.

```
df[0:2]
```

	name	species	YOB
0	Grumpy Cat	cat	2004
1	Garfi the Angry Cat	cat	2012

Slicing a pandas' DataFrame - rows

Select one or more rows using **.loc function** (**string-based**):

```
Df.loc[0:2,["name","species"]]
```

	name	species
0	Grumpy Cat	cat
1	Garfi the Angry Cat	cat

Slicing a pandas' DataFrame - rows

Select one or more rows using **.iloc function** (**position-based**):

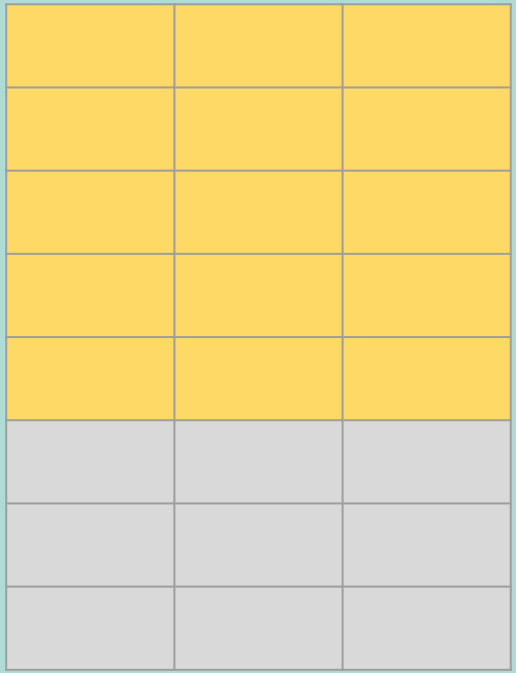
```
Df.iloc[[0,2],:]
```

	name	species	YOB
0	Grumpy Cat	cat	2004
2	Colonel Meow	cat	2011

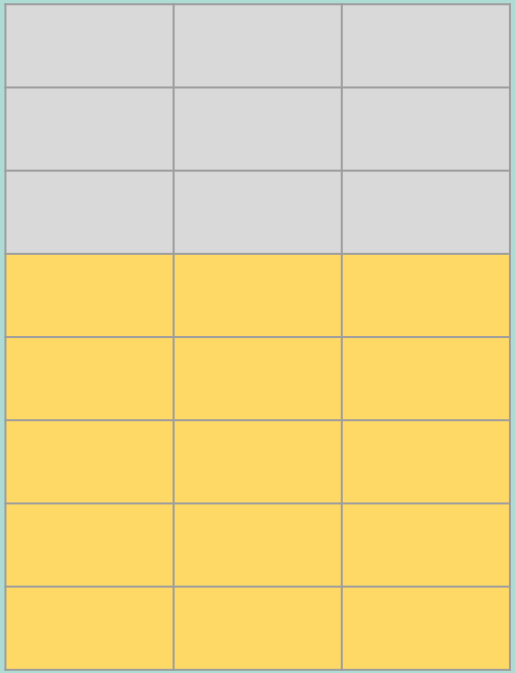
There are other ways to slice and subset dataframes, check the documentation

Pandas: some basic functions

.head( ): preview first 5 rows

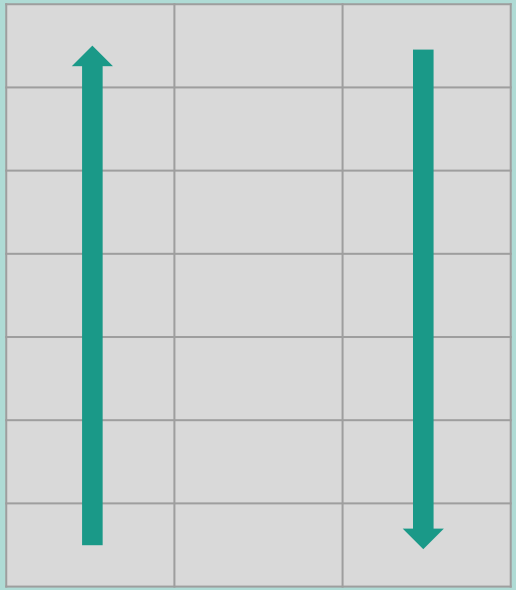


.tail( ): preview last 5 rows

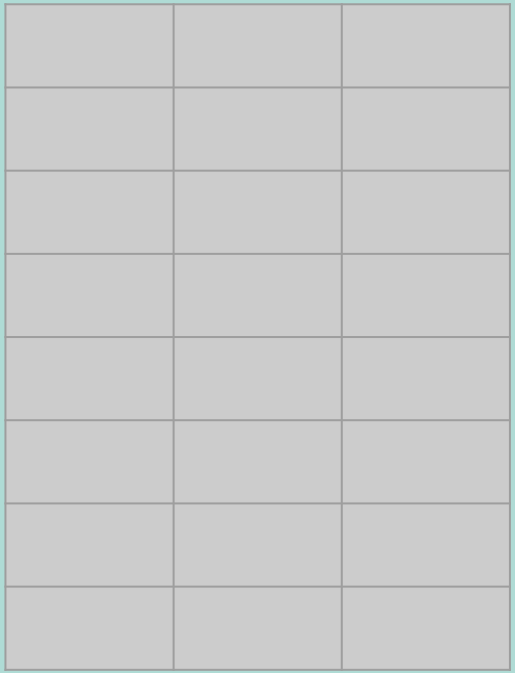


Pandas: some basic functions

sort_values(by=[‘column’]):
sorting by values



.shape: number of rows and columns



(8, 3)

Pandas: some basic functions

.dtypes: data type of each column

```
name        object
species     object
YOB          int64
dtype: object
```

.info( ): summary of dataframe

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   name     3 non-null      object
 1   species  3 non-null      object
 2   YOB      3 non-null      int64
dtypes: int64(1), object(2)
memory usage: 200.0+ bytes
```

Pandas: Common functions

Data Importing & Exporting	pd.read_csv()	Import data from a CSV file into a DataFrame.
	df.to_csv()	Export DataFrame to CSV file.
Data Inspection	df.head()	Return the first n rows.
	df.tail()	Return the last n rows.
	df.info()	Get a concise summary of the DataFrame.
	df.describe()	Generate descriptive statistics.
Data Cleaning	df.dropna()	Remove missing values.
	df.fillna()	Fill missing values.
	df.drop()	Drop specified labels from rows or columns.
	df.duplicated()	Check for duplicate rows.
	df.drop_duplicates()	Remove duplicate rows.

Also, be aware of the **syntax**: some functions require parentheses at the end, like .info( ), some don't, like .dtypes.

Pandas: Common functions

Data Manipulation	df.set_index()	Set the DataFrame index using existing columns.
	df.reset_index()	Reset the index of the DataFrame.
	df.rename()	Alter axes labels.
	df.sort_values()	Sort by the values along either axis.
	df.groupby()	Group DataFrame using a mapper or by a Series of columns.
	df.merge()	Merge DataFrame or named Series objects with a database-style join.

Also, be aware of the **syntax**.
Some functions require parentheses at the end, like df.info( ) and some don't, like df.dtypes

Pandas: some basic functions

There are **many, many more** built-in functions.
Check the documentation for more details.

Also, be aware of the **syntax**: some functions require parentheses at the end, like .info( ), some don't, like .dtypes.

Always read the documentation!