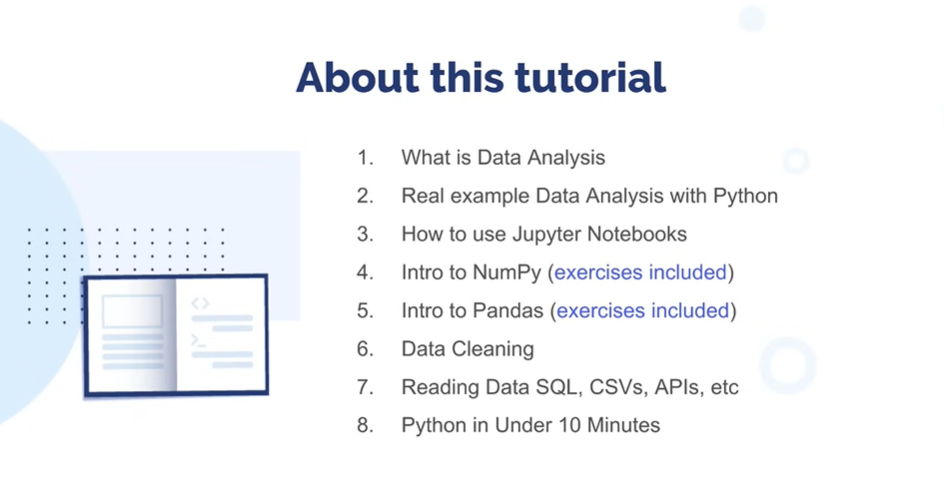
**TUTORIAL 1:**

What is Data Analysis?



**What is data Analysis?**

A process of *inspecting, cleansing, transforming and modeling* data with the goal of discovering useful information, informing conclusion and supporting decision-making.

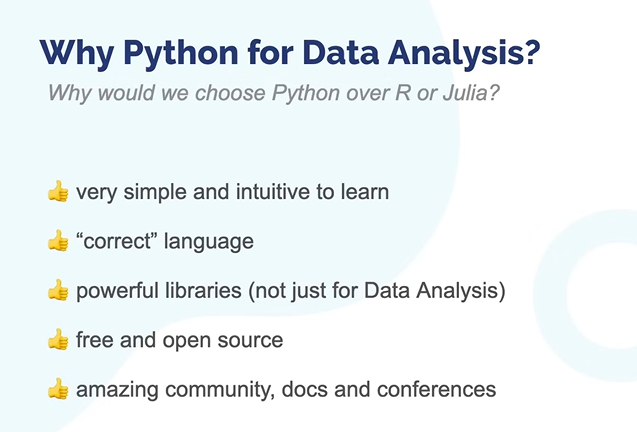
**Data** – will be transformed into information. This can be huge sales from any market

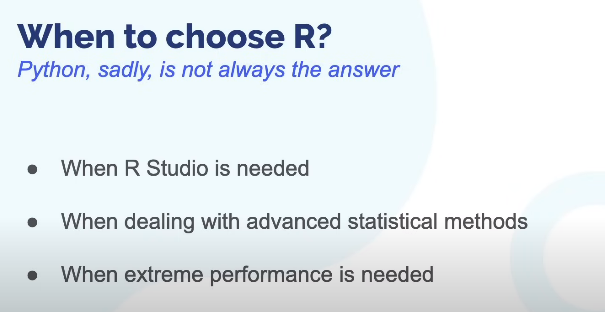
**Information** – is specific category sales or something like this.

**Data Analysis Tools**

**Auto managed closed tools** – tools we can buy and start using out of the box (*Excel, Tableau, etc*.). These tools are easy to learn. However they are limited and expensive.

**Programming languages** – are not sold by an individual vendor, they are combination of open source libraries and products (*Python, R, Julia*). These tools are open source, free or cheap. However, they can be extremely powerful. Learning programming language is not easy as well, we should learn everything step by step and with ambition and patience.

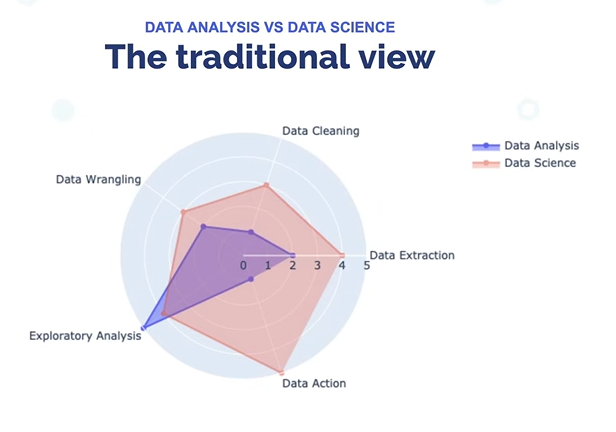


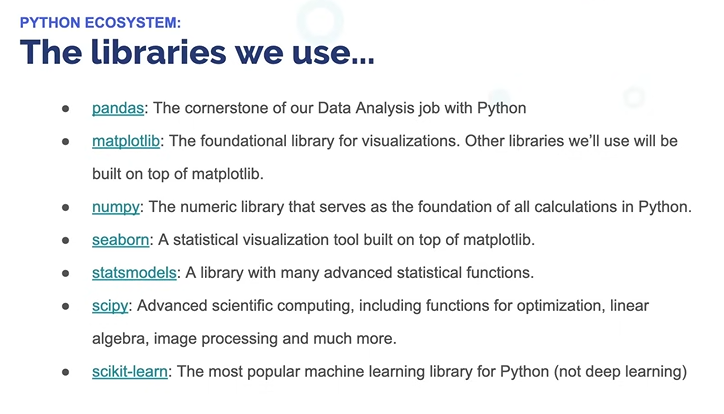


**The Data Analysis Process**

1. **Data Extraction (where we get the data):** SQL, Scrapping, File formats (CSV, JSON, XML), Consulting APIs, Buying data, Distributed databases
2. **Data Cleaning** **(after collecting we need to clean it):** Missing values and empty data, Data imputation, incorrect types, invalid values, outliers or non-relevant data, statistical sanitization
3. **Data Wrangling (we’ll need to rearrange and reshape data for analyzing):** Transforming fields, merging tables, and combining data from multiple sources
4. **Analysis**: Visualization and representation, building statistical models, correlation and causation analysis, statistical analysis is important
5. **Action**: implementing ML models for DS projects, building reports for DA projects

**Note**: this process is not linear in real life, sometimes we jump back and vice versa.





How to use Jupyter notebooks

In DS projects Jupyter Notebook is used by approximately 90% of Data Scientists. JupyterLab is an evolution of Jupyter Notebook. Difference is JupyterLab has a nicer interface for users.

It is a fully featured Python interpreter. Jupyter Notebook is a collection of **cells**. Cell can either be a **python code** or a **Markdown to format and create text**. It also can be raw, but we don’t use them.

Pressing operations/shortcuts in Jupyter. Note that they work only when cell is not in coding/edition mode.

**A** – a cell above

**D+D** – delete a cell

**B** – a cell below

**Esc** – switching from editing mode to command mode

**Return (enter below backspace)** – switching from command mode to editing mode

**Arrow keys** – moving up and down

**M** – markdown

**Y** – Python

**Ctrl + enter** – executing cell

**Shift + enter** – executing cell and switching prompt to the following one

**Z** – for undoing

**X** – for cutting cell

**C** – copying cell

Visualization and showing what you do is and advantage of Jupyter Notebook. We can get public APIs in this tool. Also after getting data, immediately showing reports and data analysis is more convenient in Jupyter notebook.

Jupyter Lab/Notebook environment works very well with CSV, JSON, XLSX, XML file types.

Note: The only way to learn Data Analysis is to practice frequently.

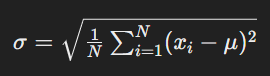
**Intro to Numpy**

One of the most important libraries in Python for data processing. It is a numeric computing library. In Python, numeric processing is slow, but when you go down to deep and work with large datasets, Python itself is not right for that. In here, Numpy is solving that.

Note: We may not be working with Numpy in real life situations **directly**. However, other libraries demand Numpy for numeric processing before them

Think we have a matrix or array named **“A”**:

A.std() means **standard deviation** of an array. It quantifies the amount of deviation or dispersion of a set of values from the mean (average) value. In other words, it tells you how much the values in a dataset differ from the mean value of the dataset.



A.var() means **variance** of an array. Variance is another measure of the spread or dispersion of data points in a dataset. It is the average of the squared differences from the mean.



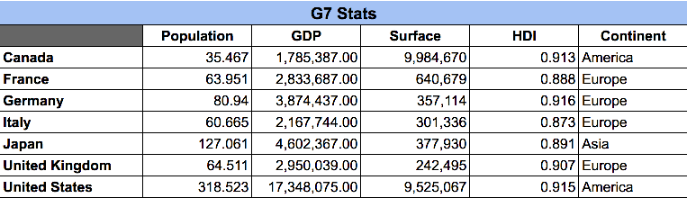
**Intro to Pandas**

Pandas helps getting data from different sources (databases, xlsx, csv), processing the data (merging, combining), visualizing data (charts/reports), simple statistical analysis

Note: Pandas has 2 data structures: Series and DataFrame.

We can say that Series look like lists. However, while looking at their indexing abilities, Series are more look alike *ordered dictionaries*.

When it comes to **DataFrames** in Pandas, they are probably the most important data structure of pandas. It is a tabular structure tightly integrated with `Series`.



Creating `DataFrame`s manually can be tedious. 99% of the time you'll be pulling the data from a Database, a csv file or the web. But still, you can create a DataFrame by specifying the columns and values:

**df = pd.DataFrame({[‘a’: 1,2,3]}, columns = ‘a’)**

DataFrame is a combination of Series, actually. They have indexes. As we can see in the table above, pandas has assigned a numeric, auto incremental index automatically to each "row" in our DataFrame.

**NOTE: Methods below always used in DataFrame:**

**info**, **size**, **shape**, **describe** (summary of statistics), **dtypes** (data types in dataframe), **dtypes.value\_counts** (dataframe values' types)

90% of operations in Pandas are immutable.

Pandas can easily read data stored in different file formats like CSV, JSON, XML or even Excel. Parsing always involves specifying the correct structure, encoding and other details. The read\_csv method reads CSV files and accepts many parameters.

NaN – not a number, which is a ‘virus’ number values. It changes everything that contacts with it.

None value is worse for of NaN, bcs NaN gives at least exceptions, however, None values if interacts, give error straightly.

Numpy also supports infinite (np.inf), and everything interacts with inf, ecomes infinite.

Np.isfinite helps to find values that are not nan or infinite.

In Pandas, Pandas manages missing values more carefully than Numpy, it is not something like ‘viruses’ in pandas. Operations will ignore them completely.

In order to drop NA values, there is **dropna()** function that comes to help us.

BTW: ALL THE OPERATIONS HERE ARE IMMUTABLE. WE ARE NOT ACTUALLY MODIFYING ANYTHING.

Sometimes instead than dropping the null values, we might need to **replace** them with some other value. This highly depends on your context and the dataset you're currently working. Sometimes *a nan can be replaced with a 0*, sometimes *it can be replaced with the mean of the sample*, and some other times you can take the closest value. Again, it depends on the context. We'll show you the different methods and mechanisms and you can then apply them to your own problem.

Fillna(method = ‘fflill’) and Fillna(method = ‘bflill’) methods, fills the missing values with elements before them and after them, respectively.

**Axis = 0, refers to the rows, axis = 1 refers to the columns.**

The methods any and all check if either there's any True value in a Series or all the values are True. They work in the same way as in Python:

**Cleaning not null values**

After dealing with many datasets I can tell you that "missing data" is not such a big deal. The best thing that can happen is to clearly see values like np.nan. The only thing you need to do is just use methods like isnull and fillna, dropna and pandas will take care of the rest.

But sometimes, you can have invalid values that are not just "missing data" (`None`, or `nan`). For example: