**Pandas and Python Tips and Tricks for Data Science and Data Analysis**

**Take your efficiency to the next level with these Pandas and Python Tricks!**

**Motivation**

This blog regroups all the Pandas and Python tricks & tips I share on a basis on my [LinkedIn](https://www.linkedin.com/in/zoumana-keita/) page. I have decided to centralize them into a single blog to help you make the most out of your learning process by easily finding what you are looking for.

The content is is divided into two main sections:

* Pandas tricks & tips are related to only Pandas.
* Python tricks & tips related to Python.

**Pandas tricks & tips**

This section provides a list of all the tricks

**1. 𝗖𝗿𝗲𝗮𝘁𝗲 𝗮 𝗻𝗲𝘄 𝗰𝗼𝗹𝘂𝗺𝗻 𝗳𝗿𝗼𝗺 𝗺𝘂𝗹𝘁𝗶𝗽𝗹𝗲 𝗰𝗼𝗹𝘂𝗺𝗻𝘀 𝗶𝗻 𝘆𝗼𝘂𝗿 𝗱𝗮𝘁𝗮𝗳𝗿𝗮𝗺𝗲.**

Performing simple arithmetic tasks such as creating a new column as the sum of two other columns can be straightforward.

🤔 But, what if you want to implement a more complex function and use it as the logic behind column creation? Here is where things can get a bit challenging.

Guess what…

✅ 𝙖𝙥𝙥𝙡𝙮 and 𝙡𝙖𝙢𝙗𝙙𝙖 can help you easily apply whatever logic to your columns using the following format:

𝙙𝙛[𝙣𝙚𝙬\_𝙘𝙤𝙡] = 𝙙𝙛.𝙖𝙥𝙥𝙡𝙮(𝙡𝙖𝙢𝙗𝙙𝙖 𝙧𝙤𝙬: 𝙛𝙪𝙣𝙘(𝙧𝙤𝙬), 𝙖𝙭𝙞𝙨=1)

where:

➡ 𝙙𝙛 is your dataframe.

➡ 𝙧𝙤𝙬 will correspond to each row in your data frame.

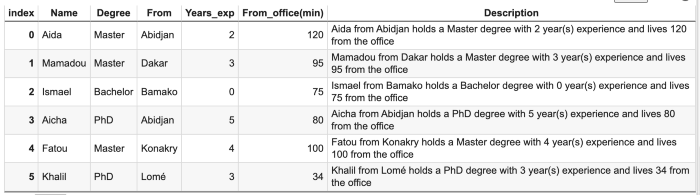
➡ 𝙛𝙪𝙣𝙘 is the function you want to apply to your data frame.

➡ 𝙖𝙭𝙞𝙨=1 to apply the function to each row in your data frame.

💡 Below is an illustration.

<https://gist.github.com/keitazoumana/07ad1c72dda94cb93a32eae061dce985#file-pandas_tricks_-multiple_cols-py>

*The `candidate\_info`* function combines each candidate’s information to create a single description column about that candidate.



Result of Pandas apply and lambda (Image by Author)

**2. Convert categorical data into numerical ones**

This process mainly can occur in the feature engineering phase. Some of its benefits are:

* the identification of outliers, invalid, and missing values in the data.
* reduction of the chance of overfitting by creating more robust models.

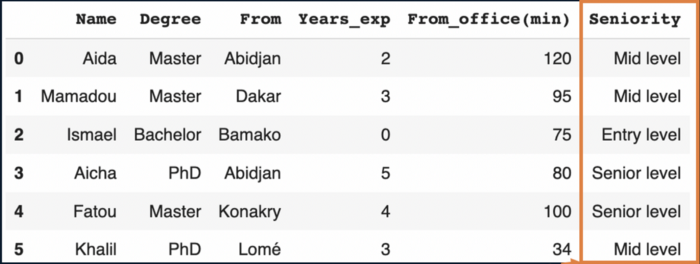
➡ Use these two functions from Pandas, depending on your need. Examples are provided in the image below.

1️⃣ .𝙘𝙪𝙩() to specifically define your bin edges.

𝙎𝙘𝙚𝙣𝙖𝙧𝙞𝙤  
Categorize candidates by expertise with respect to their number of experience, where:

* *Entry level: 0–1 year*
* *Mid-level: 2–3 years*
* *Senior level: 4–5 years*

<https://gist.github.com/keitazoumana/6af0241aad20d60cadc6b221264b7e46#file-cut_scenario-py>

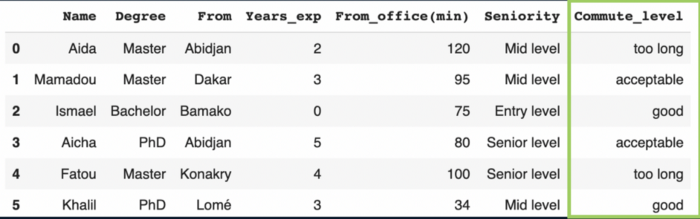


Result of the .cut function (Image by Author)

2️⃣ .𝙦𝙘𝙪𝙩() to divide your data into equal-sized bins.  
It uses the underlying percentiles of the distribution of the data, rather than the edges of the bins.

𝙎𝙘𝙚𝙣𝙖𝙧𝙞𝙤: categorize the commute time of the candidates into 𝙜𝙤𝙤𝙙, 𝙖𝙘𝙘𝙚𝙥𝙩𝙖𝙗𝙡𝙚, or 𝙩𝙤𝙤 𝙡𝙤𝙣𝙜.

<https://gist.github.com/keitazoumana/194f349c323b714a982eb53778aeb41b#file-qcut_scenario-py>



Result of the .qcut function (Image by Author)

𝙆𝙚𝙚𝙥 𝙞𝙣 𝙢𝙞𝙣𝙙 💡

* When using .𝙘𝙪𝙩(): a number of bins = number of labels + 1.
* When using .𝙦𝙘𝙪𝙩(): a number of bins = number of labels.
* With .𝙘𝙪𝙩(): set 𝙞𝙣𝙘𝙡𝙪𝙙𝙚\_𝙡𝙤𝙬𝙚𝙨𝙩=𝙏𝙧𝙪𝙚, otherwise, the lowest value will be converted to NaN.

**3. Select rows from a Pandas Dataframe based on column(s) values**

➡ use .𝙦𝙪𝙚𝙧𝙮() function by specifying the filter condition.

➡ the filter expression can contain any operators (<, >, ==, !=, etc.)

➡ use the @̷ sign to use a variable in the expression.

https://gist.github.com/keitazoumana/a7e7404afe62eb15cf6cef14724377da#file-filter\_examples-py



Select rows from a Pandas Dataframe based on column(s) values (Image by Author)

**4. Deal with zip files**

Sometimes it can be efficient to read and write .zip files without extracting them from your local disk. Below is an illustration.

<https://gist.github.com/keitazoumana/bf95d27bc9cf2d82a3d0d77fdad4ab9c#file-pandas_zip_files-py>

**5. Select 𝗮 𝘀𝘂𝗯𝘀𝗲𝘁 𝗼𝗳 𝘆𝗼𝘂𝗿 𝗣𝗮𝗻𝗱𝗮𝘀 𝗱𝗮𝘁𝗮𝗳𝗿𝗮𝗺𝗲 𝘄𝗶𝘁𝗵 𝘀𝗽𝗲𝗰𝗶𝗳𝗶𝗰 𝗰𝗼𝗹𝘂𝗺𝗻 𝘁𝘆𝗽𝗲𝘀**

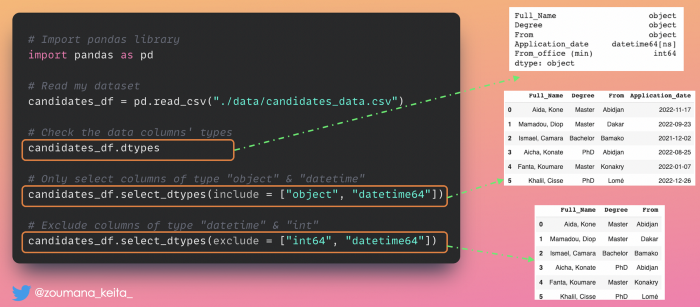
You can use the 𝙨𝙚𝙡𝙚𝙘𝙩\_𝙙𝙩𝙮𝙥𝙚𝙨 function. It takes two main parameters: 𝚒𝚗𝚌𝚕𝚞𝚍𝚎 𝚊𝚗𝚍 𝚎𝚡𝚌𝚕𝚞𝚍𝚎.

* 𝚍𝚏.𝚜𝚎𝚕𝚎𝚌𝚝\_𝚍𝚝𝚢𝚙𝚎𝚜(𝚒𝚗𝚌𝚕𝚞𝚍𝚎 = [‘𝚝𝚢𝚙𝚎\_𝟷’, ‘𝚝𝚢𝚙𝚎\_𝟸’, … ‘𝚝𝚢𝚙𝚎\_𝚗’]) means I want the subset of my data frame WITH columns of 𝚝𝚢𝚙𝚎\_𝟷, 𝚝𝚢𝚙𝚎\_𝟸,…, 𝚝𝚢𝚙𝚎\_𝚗.
* 𝚍𝚏.𝚜𝚎𝚕𝚎𝚌𝚝\_𝚍𝚝𝚢𝚙𝚎𝚜(𝚎𝚡𝚌𝚕𝚞𝚍𝚎 = [‘𝚝𝚢𝚙𝚎\_𝟷’, ‘𝚝𝚢𝚙𝚎\_𝟸’, … ‘𝚝𝚢𝚙𝚎\_𝚗’]) means I want the subset of my data frame WITHOUT columns of 𝚝𝚢𝚙𝚎\_𝟷, 𝚝𝚢𝚙𝚎\_𝟸,…, 𝚝𝚢𝚙𝚎\_𝚗.

✨ Below is an illustration

<https://gist.github.com/keitazoumana/2ea1f5d53a125655e8806b73392e7e6c#file-select_subset_column_types-py>

[select\_subset\_column\_types.py](https://gist.github.com/keitazoumana/2ea1f5d53a125655e8806b73392e7e6c#file-select_subset_column_types-py)



Columns subset selection (Image by Author)

**6. Remove comments from Pandas dataframe column**

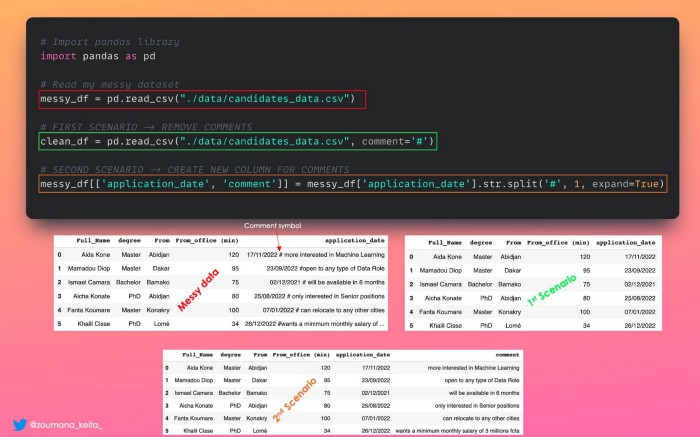
Imagine that I want clean this data (candidates.csv) by removing comments from the application date column. This can be done on the fly while loading your pandas dataframe using the 𝙘𝙤𝙢𝙢𝙚𝙣𝙩 parameter as follow:

➡ 𝚌𝚕𝚎𝚊𝚗\_𝚍𝚊𝚝𝚊 = 𝚙𝚍.𝚛𝚎𝚊𝚍\_𝚌𝚜𝚟(𝚙𝚊𝚝𝚑\_𝚝𝚘\_𝚍𝚊𝚝𝚊, 𝙘𝙤𝙢𝙢𝙚𝙣𝙩=’𝚜𝚢𝚖𝚋𝚘𝚕’)

In my case, 𝙘𝙤𝙢𝙢𝙚𝙣𝙩=’#’ but it could be any other character (|, /, etc.) depending on your case. An illustration is the first scenario.

✋🏽 Wait, what if I want to create a new column for those comments and still remove them from the application date column? An illustration is the second scenario.

<https://gist.github.com/keitazoumana/58ce6adc553527195516b11879e5f97e#file-pandas_remove_comments-py>



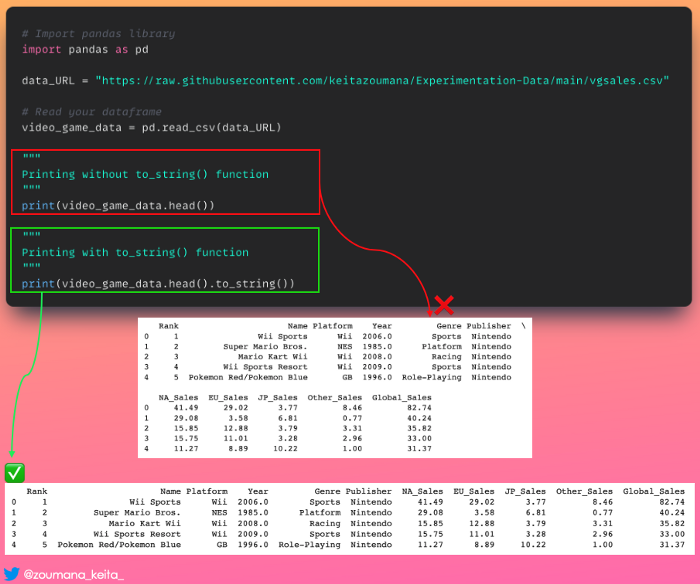
Remove comments from pandas dataframe (Image by Author)

**7. Print Pandas dataframe in Tabular format from consol**

❌ No, the application of the 𝚙𝚛𝚒𝚗𝚝() function to a pandas data frame does not always render an output that is easy to read, especially for data frames with multiple columns.

✅ If you want to get a nice console-friendly tabular output  
Use the .𝚝𝚘\_𝚜𝚝𝚛𝚒𝚗𝚐() function as illustrated below.

<https://gist.github.com/keitazoumana/e8c94a17e303207501f7593f9188c6da#file-pandas_to_string-py>



**8. Highlight data points in Pandas**

Applying colors to a pandas data frame can be a good way to emphasize certain data points for quick analysis.

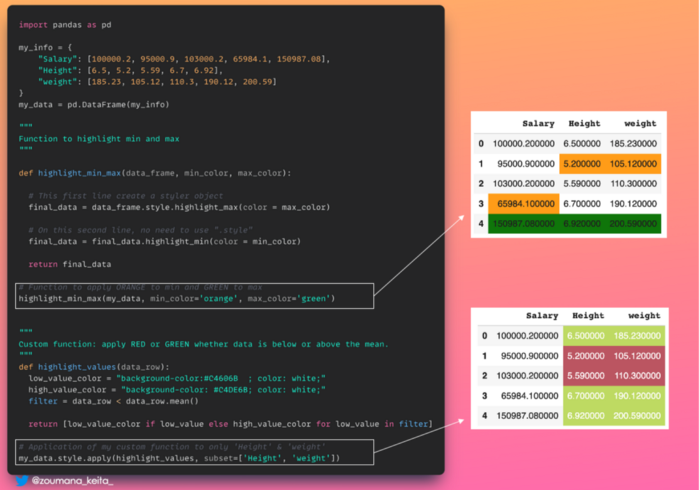
✅ This is where 𝚙𝚊𝚗𝚍𝚊𝚜.𝚜𝚝𝚢𝚕𝚎 module comes in handy. It has many features, but is not limited to the followings:

✨ 𝚍𝚏.𝚜𝚝𝚢𝚕𝚎.𝚑𝚒𝚐𝚑𝚕𝚒𝚐𝚑𝚝\_𝚖𝚊𝚡() to assign a color to the maximum value of each column.

✨ 𝚍𝚏.𝚜𝚝𝚢𝚕𝚎.𝚑𝚒𝚐𝚑𝚕𝚒𝚐𝚑𝚝\_𝚖in() to assign a color to the minimum value of each column.

✨ 𝚍𝚏.𝚜𝚝𝚢𝚕𝚎.𝚊𝚙𝚙𝚕𝚢(𝚖𝚢\_𝚌𝚞𝚜𝚝𝚘𝚖\_𝚏𝚞𝚗𝚌𝚝𝚒𝚘𝚗) to apply your custom function to your data frame.

<https://gist.github.com/keitazoumana/6cc7fdb7c4460b09adaeaf431ecb7957#file-highlight_pandas_datapoints-py>



Highlight data points in Pandas (Image by Author)

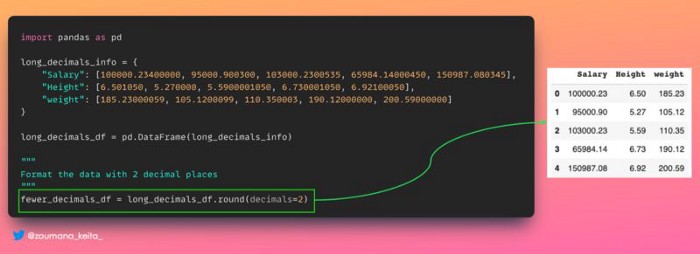
**9. Reduce decimal points in your data**

Sometimes, very long decimal values in your data set do not provide significant information and can be painful 🤯 to look at.

So, you might want to convert your data to about 2 to 3 decimal points to facilitate your analysis.

✅ This is something you can perform using the 𝚙𝚊𝚗𝚍𝚊𝚜.𝙳𝚊𝚝𝚊𝙵𝚛𝚊𝚖𝚎.𝚛𝚘𝚞𝚗𝚍() function as illustrated below.

<https://gist.github.com/keitazoumana/cc0b5c43a78ef178ce4a2ea192a1462f#file-long_to_few_decimals-py>

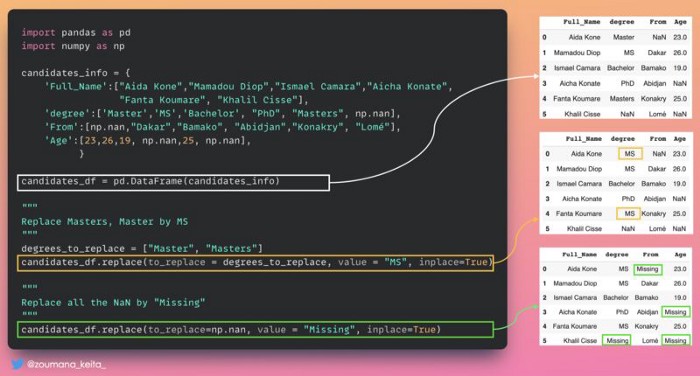


Reduce decimal points in your data (Image by Author)

**10. Replace some values in your data frame**

You might want to replace some information in your data frame to keep it as up-to-date as possible.

✅ This can be achieved using the Pandas 𝚍𝚊𝚝𝚊𝚏𝚛𝚊𝚖𝚎.𝚛𝚎𝚙𝚕𝚊𝚌𝚎() function as illustrated below.



Replace some values in your data frame (Image by Author)

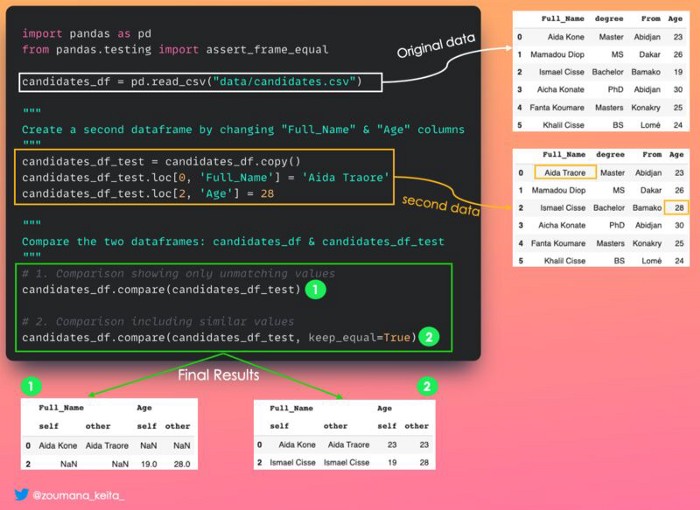
**11. Compare two data frames and get their differences**

Sometimes, when comparing two pandas data frames, not only do you want to know if they are equivalent, but also where the difference lies if they are not equivalent.

✅ This is where the .𝚌𝚘𝚖𝚙𝚊𝚛𝚎() function comes in handy.

✨ It generates a data frame showing columns with differences side by side. Its shape is different from (0, 0) only if the two data being compared are the same.

✨ If you want to show values that are equal, set the 𝚔𝚎𝚎𝚙\_𝚎𝚚𝚞𝚊𝚕 parameter to 𝚃𝚛𝚞𝚎. Otherwise, they are shown as 𝙽𝚊𝙽.



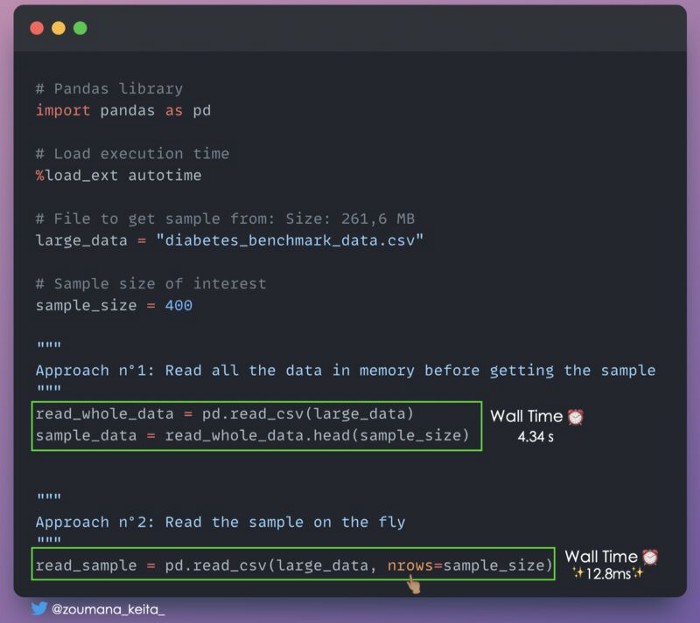
Compare two data frames and get their differences (Image by Author)

**12. Get a subset of a very large dataset for quick analysis**

Sometimes, we just need a subset of a very large dataset for quick analysis. One of the approaches could be to read the whole data in memory before getting your sample.

This can require a lot of memory depending on how big your data is. Also, it can take significant time to read your data.

✅ You can use 𝚗𝚛𝚘𝚠𝚜 parameter in the pandas 𝚛𝚎𝚊𝚍\_𝚌𝚜𝚟() function by specifying the number of rows you want.



Get a subset of a very large dataset for quick analysis (Image by Author)

**13. Transform your data frame from a wide to a long format**

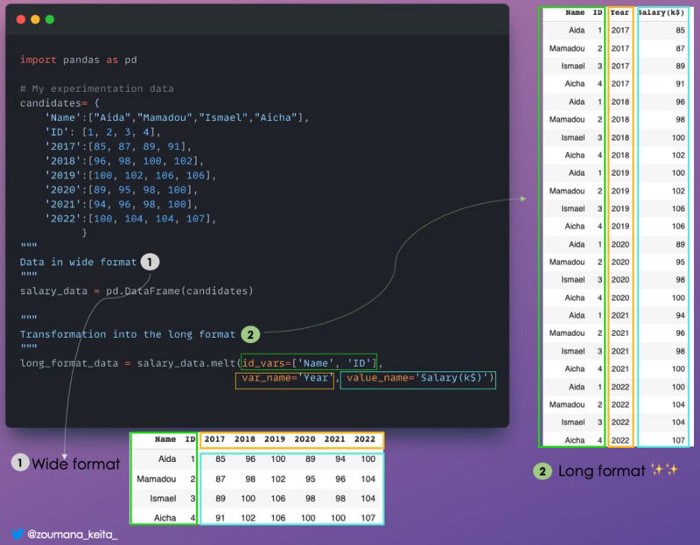
Sometimes it can be useful 𝚝𝚛𝚊𝚗𝚜𝚏𝚘𝚛𝚖 𝚢𝚘𝚞𝚛 𝚍𝚊𝚝𝚊𝚏𝚛𝚊𝚖𝚎 𝚏𝚛𝚘𝚖 𝚊 𝚠𝚒𝚍𝚎 𝚝𝚘 𝚊 𝚕𝚘𝚗𝚐 𝚏𝚘𝚛𝚖𝚊𝚝 which is more flexible for better analysis, especially when dealing with time series data.

* 𝙒𝙝𝙖𝙩 𝙙𝙤 𝙮𝙤𝙪 𝙢𝙚𝙖𝙣 𝙗𝙮 𝙬𝙞𝙙𝙚 & 𝙡𝙤𝙣𝙜?

✨ Wide format is when you have a lot of columns.  
✨ Long format on the other side is when you have a lot of rows.

✅ 𝙿𝚊𝚗𝚍𝚊𝚜.𝚖𝚎𝚕𝚝() is a perfect candidate for this task.

Below is an illustration



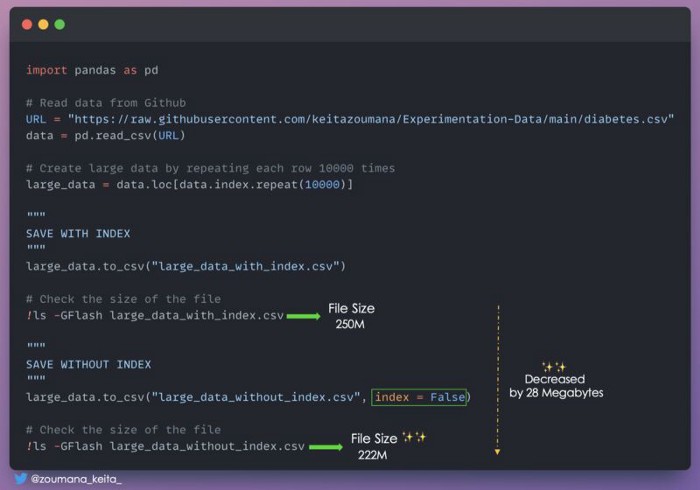
Transform your data frame from a wide to a long format (Image by Author)

**14. Reduce the size of your Pandas data frame by ignoring the index**

Do you know that you can reduce the size of your Pandas data frame by ignoring the index when saving it?

✅ Something like 𝚒𝚗𝚍𝚎𝚡 = 𝙵𝚊𝚕𝚜𝚎 when saving the file.

Below is an illustration.



Reduce the size of your Pandas data frame by ignoring the index (Image by Author)

**15. Parquet instead of CSV**

Very often, I don’t manually look 👀 at the content of a CSV or Excel file that will be used by Pandas for further analysis.

If that’s your case, maybe you should not use .CSV anymore and think of a better option.

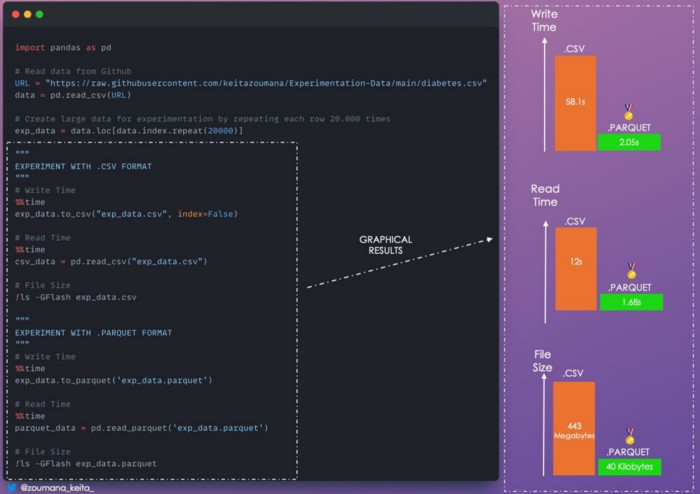
Especially if you are only concerned about

✨ Processing speed

✨ Speed in saving and loading

✨ Disk space occupied by the data frame

✅ In that case, .𝙥𝙖𝙧𝙦𝙪𝙚𝙩 format is your best option as illustrated below.



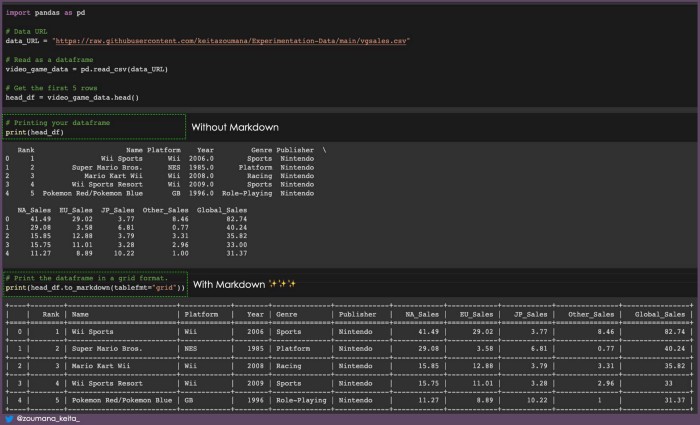
Parquet instead of CSV (Image by Author)

**16. Transform your data frame into a markdown**

It is always better to print your data frame in a way that makes it easier to understand.

✅ One way of doing that is to render it in a markdown format using the .𝚝𝚘\_𝚖𝚊𝚛𝚔𝚍𝚘𝚠𝚗() function.

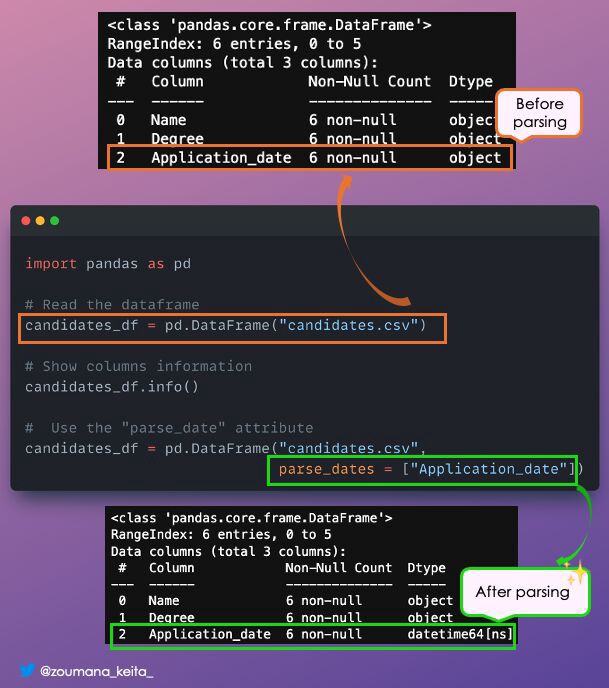
💡 Below is an illustration



**17. Format Date Time column**

When loading Pandas dataframes, date columns are represented as 𝗼𝗯𝗷𝗲𝗰𝘁 by default, which is not ❌ the correct date format.

✅ You can specify the target column in the 𝗽𝗮𝗿𝘀𝗲\_𝗱𝗮𝘁𝗲𝘀 argument to get the correct column type.



DateTime Formating

**Python tips and tricks**

**1. Create a progress bar with tqdm and rich**

Using the progress bar is beneficial when you want to have a visual status of a given task.

#!pip -q install rich  
from rich.progress import track  
from tqdm import tqdm  
import time

Implement the callback function

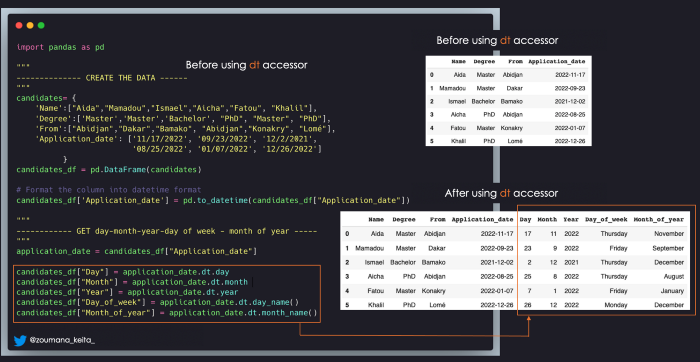
def compute\_double(x):  
 return 2\*x

Create the progress bars

rich progress bar implementation

tqdm progress bar implementation

**2. Get day, month, year, day of the week, the month of the year**



Get day, month, year, day of the week, the month of the year (Image by author)

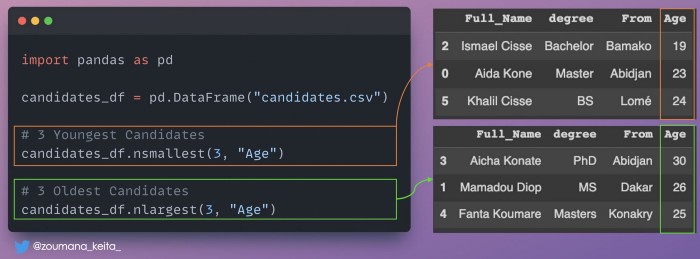
**3. Smallest and largest values of a column**

If you want to get the rows with the largest or lowest values for a given column, you can use the following functions:

✨ 𝚍𝚏.𝚗𝚕𝚊𝚛𝚐𝚎𝚜𝚝(𝙽, “𝙲𝚘𝚕\_𝙽𝚊𝚖𝚎”) → top 𝙽 rows based on 𝙲𝚘𝚕\_𝙽𝚊𝚖𝚎

✨ 𝚍𝚏.𝚗𝚜𝚖𝚊𝚕𝚕𝚎𝚜𝚝(𝙽, “𝙲𝚘𝚕\_𝙽𝚊𝚖𝚎”) → 𝙽 smallest rows based on 𝙲𝚘𝚕\_𝙽𝚊𝚖𝚎

✨ 𝙲𝚘𝚕\_𝙽𝚊𝚖𝚎 is the name of the column you are interested in.



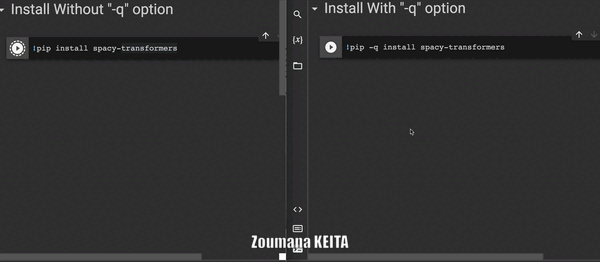
Smallest and largest values illustration (Image by Author)

**4. Ignore the log output of the pip install command**

Sometimes when installing a library from your jupyter notebook, you might not want to have all the details about the installation process generated by the default 𝚙𝚒𝚙 𝚒𝚗𝚜𝚝𝚊𝚕𝚕 command.

✅ You can specify the -q or — quiet option to get rid of that information.

Below is an illustration 💡



pip install illustration (Animation by Author)

**5. Run multiple commands in a single notebook cell**

The exclamation mark ‘!’ is essential to successfully run a shell command from your Jupyter notebook.

However, this approach can be quite repetitive 🔂 when dealing with multiple commands or a very long and complicated one.

✅ A better way to tackle this issue is to use the %%𝐛𝐚𝐬𝐡 expression at the beginning of your notebook cell.

💡 Below is an illustration

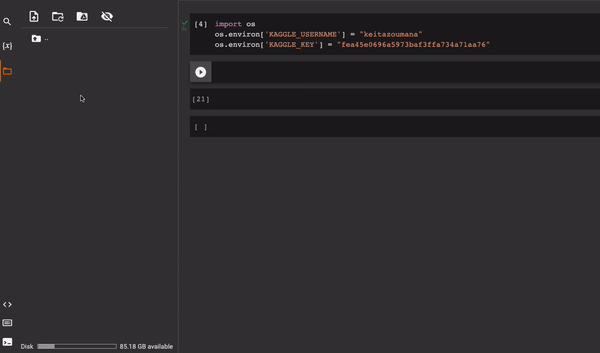


Illustration of %%bash statement (Animation by Autor)

**6. Virtual environment.**

A Data Science project can involve multiple dependencies, and dealing with all of them can be a bit annoying. 🤯

✨ A good practice is to organize your project in a way that it can be easily shared with your team members and reproduced with the least amount of effort.

✅ One way of doing this is to use virtual environments.

⚙️ 𝗖𝗿𝗲𝗮𝘁𝗲 𝘃𝗶𝗿𝘁𝘂𝗮𝗹 𝗲𝗻𝘃𝗶𝗿𝗼𝗻𝗺𝗲𝗻𝘁 𝗮𝗻𝗱 𝗶𝗻𝘀𝘁𝗮𝗹𝗹 𝗹𝗶𝗯𝗿𝗮𝗿𝗶𝗲𝘀.

→ Install the virtual environment module.  
𝚙𝚒𝚙 𝚒𝚗𝚜𝚝𝚊𝚕𝚕 𝚟𝚒𝚛𝚝𝚞𝚊𝚕𝚎𝚗𝚟

→ Create your environment by giving a meaningful name.  
𝚟𝚒𝚛𝚝𝚞𝚊𝚕𝚎𝚗𝚟 [𝚢𝚘𝚞𝚛\_𝚎𝚗𝚟𝚒𝚛𝚘𝚗𝚖𝚎𝚗𝚝\_𝚗𝚊𝚖𝚎]

→ Activate your environment.  
𝚜𝚘𝚞𝚛𝚌𝚎 [𝚢𝚘𝚞𝚛\_𝚎𝚗𝚟𝚒𝚛𝚘𝚗𝚖𝚎𝚗𝚝\_𝚗𝚊𝚖𝚎]/𝚋𝚒𝚗/𝚊𝚌𝚝𝚒𝚟𝚊𝚝𝚎

→ Start installing the dependencies for your project.  
𝚙𝚒𝚙 𝚒𝚗𝚜𝚝𝚊𝚕𝚕 𝚙𝚊𝚗𝚍𝚊𝚜  
…

All this is great 👏🏼, BUT… the virtual environment you just created is local to your machine😏.

𝙒𝙝𝙖𝙩 𝙩𝙤 𝙙𝙤?🤷🏻‍♂️

💡 You need to permanently save those dependencies in order to share them with others using this command:

→ 𝚙𝚒𝚙 𝚏𝚛𝚎𝚎𝚣𝚎 > 𝚛𝚎𝚚𝚞𝚒𝚛𝚎𝚖𝚎𝚗𝚝𝚜.𝚝𝚡𝚝

This will create 𝚛𝚎𝚚𝚞𝚒𝚛𝚎𝚖𝚎𝚗𝚝𝚜.𝚝𝚡𝚝 file containing your project dependencies.

🔚 Finally, anyone can install the exact same dependencies by running this command:  
→ 𝚙𝚒𝚙 𝚒𝚗𝚜𝚝𝚊𝚕𝚕 -𝚛 𝚛𝚎𝚚𝚞𝚒𝚛𝚎𝚖𝚎𝚗𝚝𝚜.𝚝𝚡𝚝

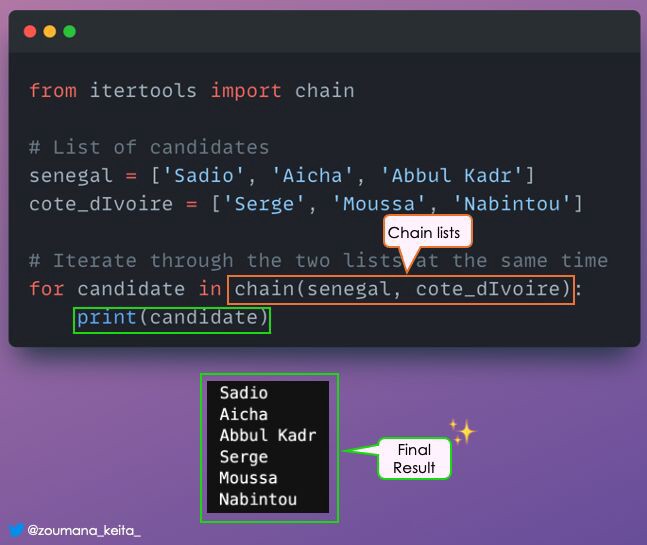
**7. Run multiple metrics at once**

Scikit learn metrics

**8. Chain multiple lists as a single sequence**

You can use a single for loop to iterate through multiple lists as a single sequence 🔂.

✅ This can be achieved using the 𝚌𝚑𝚊𝚒𝚗() ⛓ function from Python 𝗶𝘁𝗲𝗿𝘁𝗼𝗼𝗹𝘀 module.

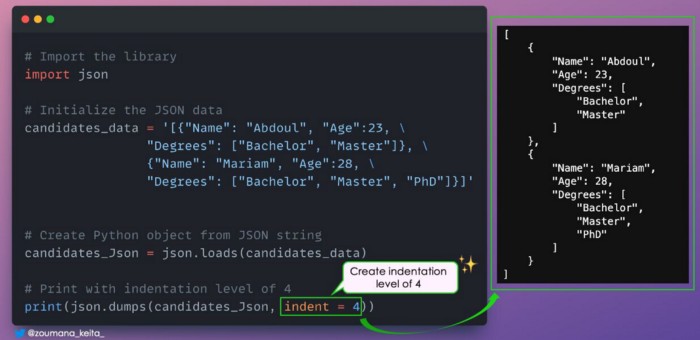


List chaining

**9. Pretty print of JSON data**

❓ Have ever wanted to print your JSON data in a correct indented format for better visualization?

✅ The indent parameter of the dumps() method can be used to specify the indentation level of your formatted string output.



Pretty print your JSON data

**Conclusion**

Thank you for reading! 🎉 🍾

I hope you found this list of Python and Pandas tricks helpful! Keep an eye on here, because the content will be maintained with more tricks on a daily basis.

Also, If you like reading my stories and wish to support my writing, consider [becoming a Medium member](https://zoumanakeita.medium.com/membership). With a $ 5-a-month commitment, you unlock unlimited access to stories on Medium.

Feel free to follow me on [Medium](https://zoumanakeita.medium.com/), [Twitter](https://twitter.com/zoumana_keita_), and [YouTube](https://www.youtube.com/channel/UC9xKdy8cz6ZuJU5FTNtM_pQ), or say Hi on [LinkedIn](https://www.linkedin.com/in/zoumana-keita/). It is always a pleasure to discuss AI, ML, Data Science, NLP, and MLOps stuff!