Photo by Andrew Neel on Unsplash

Motivation

This blog regroups all the Pandas and Python tricks & tips I share on a basis on my LinkedIn 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 divided into two main sections:

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

If you are more of a video person, you can start watching my series about these tricks on my YouTube channel for more interactivity. Each video covers about two or three tricks at a time.

...

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.

```python
import pandas as pd

# Create the dataframe
candidates= {
    'Name':["Aida","Mamadou","Ismael","Aicha","Fatou", "Khalil"],
    'Degree':['Master','Master','Bachelor', "PhD", "Master", "PhD"],
    'From':["Abidjan","Dakar","Bamako", "Abidjan","Konakry", "Lomé"],
    'Years_exp': [2, 3, 0, 5, 4, 3],
    'From_office(min)': [120, 95, 75, 80, 100, 34]
         }
candidates_df = pd.DataFrame(candidates)

"""
----------------My custom function-------------------
"""
def candidate_info(row):

  # Select columns of interest
  name = row.Name
  is_from = row.From
  year_exp = row.Years_exp
  degree = row.Degree
  from_office = row["From_office(min)"]

  # Generate the description from previous variables
  info = f"""{name} from {is_from} holds a {degree} degree
              with {year_exp} year(s) experience
              and lives {from_office} from the office"""

  return info

"""
-------Application of the function to the data ------
"""
candidates_df["Description"] = candidates_df.apply(lambda row: candidate_info(row),
axis=1)
```

pandas_tricks_.multiple_cols.py hosted with ❤️ by GitHub view raw

The `candidate_info` function combines each candidate's information to create a single description column about that candidate.

index	Name	Degree	From	Years_exp	From_office(min)	Description
0	Aida	Master	Abidjan	2	120	Aida from Abidjan holds a Master degree with 2 year(s) experience and lives 120 from the office
1	Mamadou	Master	Dakar	3	95	Mamadou from Dakar holds a Master degree with 3 year(s) experience and lives 95 from the office
2	Ismael	Bachelor	Bamako	0	75	Ismael from Bamako holds a Bachelor degree with 0 year(s) experience and lives 75 from the office
3	Aicha	PhD	Abidjan	5	80	Aicha from Abidjan holds a PhD degree with 5 year(s) experience and lives 80 from the office
4	Fatou	Master	Konakry	4	100	Fatou from Konakry holds a Master degree with 4 year(s) experience and lives 100 from the office
5	Khalil	PhD	Lomé	3	34	Khalil from Lomé holds a PhD degree with 3 year(s) experience and lives 34 from the office

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*

```
seniority = ['Entry level', 'Mid level', 'Senior level']
seniority_bins = [0, 1, 3, 5]
candidates_df['Seniority'] = pd.cut(candidates_df['Years_exp'],
                                    bins=seniority_bins,
                                    labels=seniority,
                                    include_lowest=True)

candidates_df
```

cut_scenario.py hosted with ❤️ by GitHub view raw

	Name	Degree	From	Years_exp	From_office(min)	Seniority
0	Aida	Master	Abidjan	2	120	Mid level
1	Mamadou	Master	Dakar	3	95	Mid level
2	Ismael	Bachelor	Bamako	0	75	Entry level
3	Aicha	PhD	Abidjan	5	80	Senior level
4	Fatou	Master	Konakry	4	100	Senior level
5	Khalil	PhD	Lomé	3	34	Mid level

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***.

```
commute_time_labels = ["good", "acceptable", "too long"]
candidates_df["Commute_level"] = pd.qcut(
                                  candidates_df["From_office(min)"],
                                  q = 3,
                                  labels=commute_time_labels
                                 )
candidates_df
```

qcut_scenario.py hosted with ❤️ by GitHub view raw

	Name	Degree	From	Years_exp	From_office(min)	Seniority	Commute_level
0	Aida	Master	Abidjan	2	120	Mid level	too long
1	Mamadou	Master	Dakar	3	95	Mid level	acceptable
2	Ismael	Bachelor	Bamako	0	75	Entry level	good
3	Aicha	PhD	Abidjan	5	80	Senior level	acceptable
4	Fatou	Master	Konakry	4	100	Senior level	too long
5	Khalil	PhD	Lomé	3	34	Mid level	good

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.

```
# Get all the candidates with a Master degree
ms_candidates = candidates_df.query("Degree == 'Master'")

# Get non bachelor candidates
no_bs_candidates = candidates_df.query("Degree != 'Bachelor'")

# Get values from list
list_locations = ["Abidjan", "Dakar"]
candiates = candidates_df.query("From in @list_locations")
```

filter_examples.py hosted with ❤ by GitHub view raw

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.

```
import pandas as pd

"""
------------ READ ZIP FILES -----------
"""
# Case 1: read a single zip file
candidate_df_unzip = pd.read_csv('candidates.csv.zip', compression='zip')

# Case 2: read a file from a folder
from zipfile import ZipFile

# Read the file from a zip folder
sales_df = pd.read_csv(ZipFile("data.zip").open('data/sales_df.csv'))


"""
------------ WRITE ZIP FILES -----------
"""
# Read data from internet
url = "https://raw.githubusercontent.com/keitazoumana/Fastapi-tutorial/master/data/spam.csv"
spam_data = pd.read_csv(url, encoding="ISO-8859-1")

# Save it as a zip file
spam_data.to_csv("spam.csv.zip", compression="zip")

# Check the files sizes
from os import path
path.getsize('spam.csv') / path.getsize('spam.csv.zip')
```

pandas_zip_files.py hosted with ❤️ by GitHub view raw

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`.

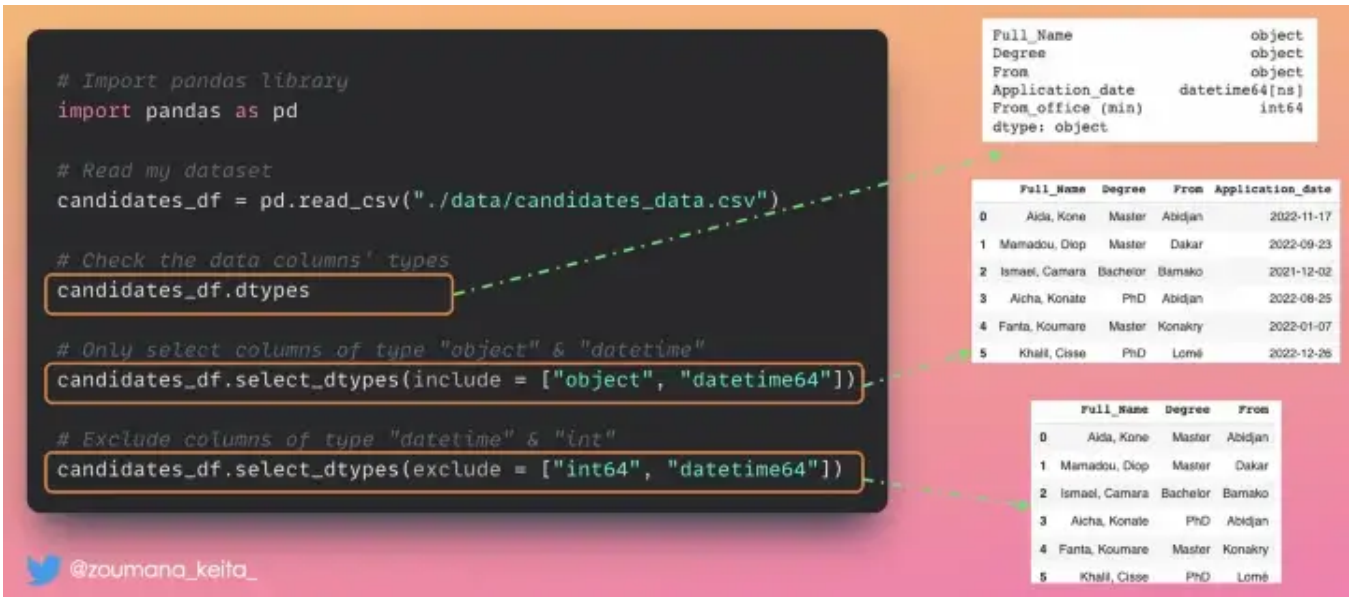
- `df.select_dtypes(exclude = ['type_1', 'type_2', ... 'type_n'])` means I want the subset of my data frame WITHOUT columns of `type_1`, `type_2`,..., `type_n`.

✨ Below is an illustration

```python
# Import pandas library
import pandas as pd

# Read my dataset
candidates_df = pd.read_csv("./data/candidates_data.csv")

# Check the data columns' types
candidates_df.dtypes

# Only select columns of type "object" & "datetime"
candidates_df.select_dtypes(include = ["object", "datetime64"])

# Exclude columns of type "datetime" & "int"
candidates_df.select_dtypes(exclude = ["int64", "datetime64"])
```

select_subset_column_types.py hosted with ❤️ by GitHub view raw

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:

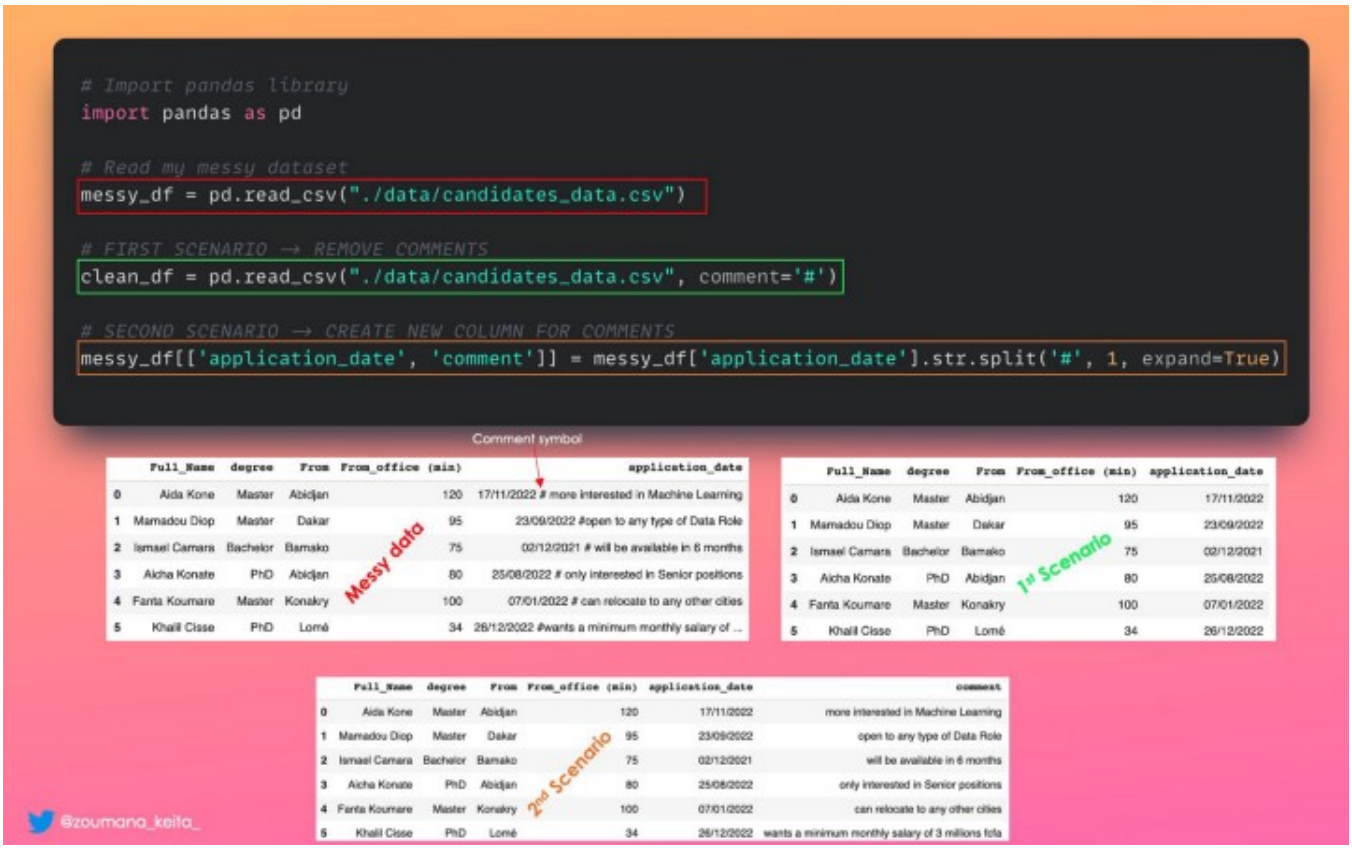
➡ `clean_data = pd.read_csv(path_to_data, comment='symbol')`

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.

```
# Import pandas library
import pandas as pd

# Read my messy dataset
messy_df = pd.read_csv("./data/candidates_data.csv")

# FIRST SCENARIO -> REMOVE COMMENTS
clean_df = pd.read_csv("./data/candidates_data.csv", comment='#')

# SECOND SCENARIO -> CREATE NEW COLUMN FOR COMMENTS
messy_df[['application_date', 'comment']] =
messy_df['application_date'].str.split('#', 1, expand=True)
```

pandas_remove_comments.py hosted with ❤️ by GitHub view raw



Remove comments from pandas dataframe (Image by Author)

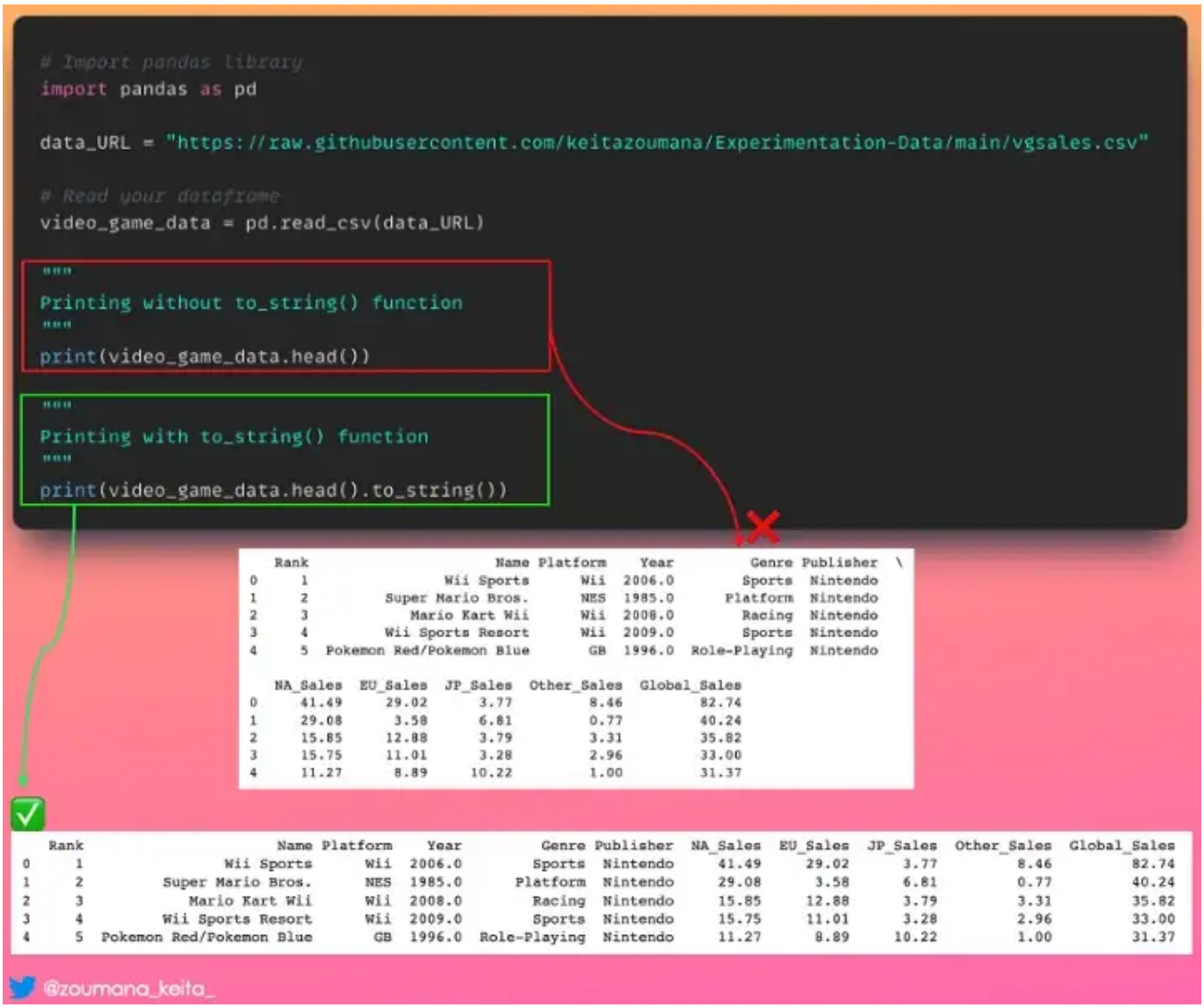
7. Print Pandas dataframe in Tabular format from consol

❌ 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.

```python
# Import pandas library
import pandas as pd

data_URL = "https://raw.githubusercontent.com/keitazoumana/Experimentation-Data/main/vgsales.csv"

# Read your dataframe
video_game_data = pd.read_csv(data_URL)

"""
Printing without to_string() function
"""
print(video_game_data.head())

"""
Printing with to_string() function
"""
print(video_game_data.head().to_string())
```

pandas_to_string.py hosted with ❤️ by GitHub view raw



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.

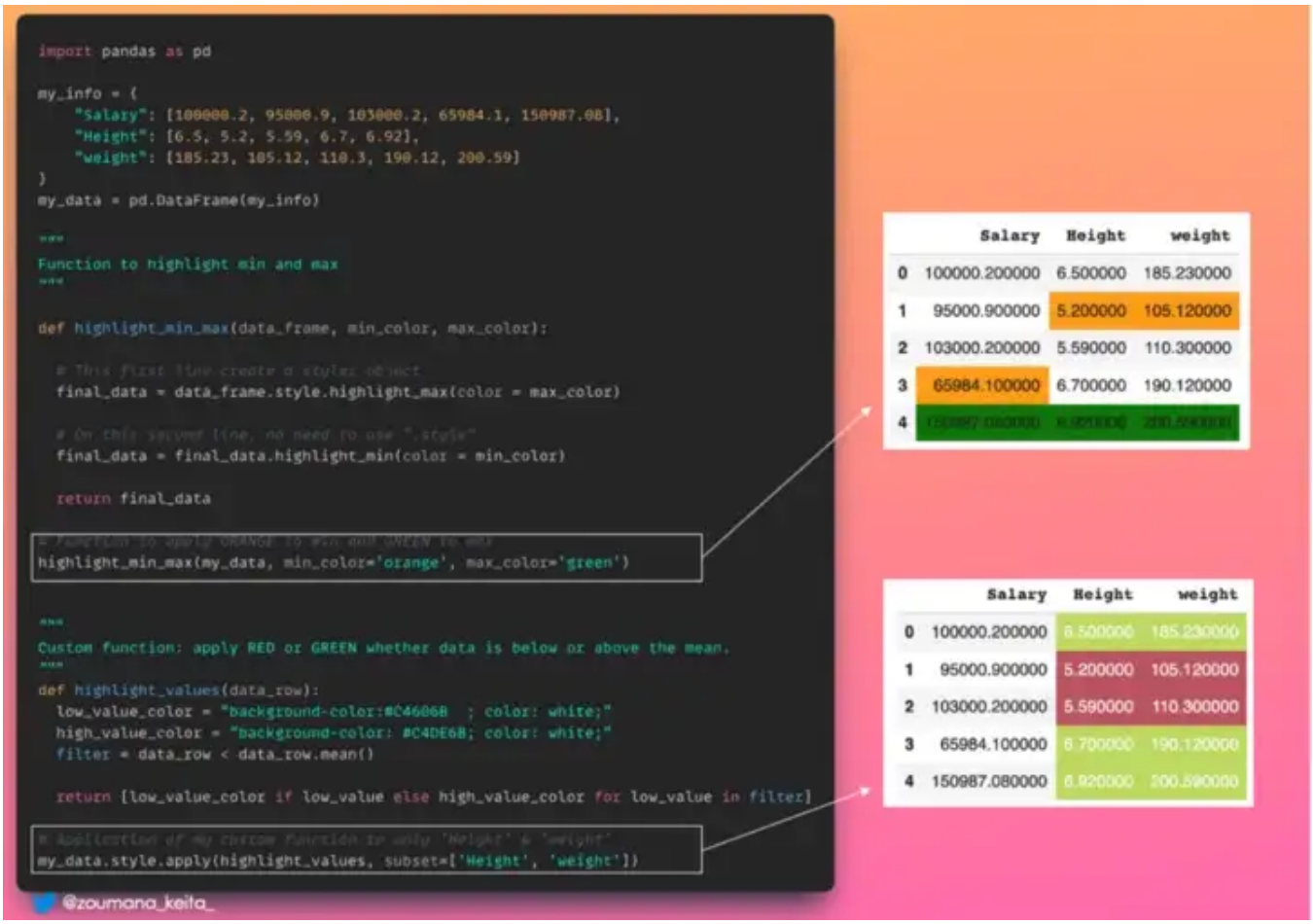
✅ This is where `pandas.style` module comes in handy. It has many features, but is not limited to the followings:

✨ `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.

```python
import pandas as pd

my_info = {
    "Salary": [100000.2, 95000.9, 103000.2, 65984.1, 150987.08],
    "Height": [6.5, 5.2, 5.59, 6.7, 6.92],
    "weight": [185.23, 105.12, 110.3, 190.12, 200.59]
}
my_data = pd.DataFrame(my_info)

"""
Function to highlight min and max
"""

def highlight_min_max(data_frame, min_color, max_color):

  # This first line create a styler object
  final_data = data_frame.style.highlight_max(color = max_color)

  # On this second line, no need to use ".style"
  final_data = final_data.highlight_min(color = min_color)

  return final_data

# Function to apply ORANGE to min and GREEN to max
highlight_min_max(my_data, min_color='orange', max_color='green')


"""
Custom function: apply RED or GREEN whether data is below or above the mean.
"""
def highlight_values(data_row):
  low_value_color = "background-color:#C4606B  ; color: white;"
  high_value_color = "background-color: #C4DE6B; color: white;"
  filter = data_row < data_row.mean()

  return [low_value_color if low_value else high_value_color for low_value in filter]

# Application of my custom function to only 'Height' & 'weight'
my_data.style.apply(highlight_values, subset=['Height', 'weight'])
```



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.

```python
long_decimals_info = {
    "Salary": [100000.23400000, 95000.900300, 103000.2300535, 65984.14000450, 150987.080345],
    "Height": [6.501050, 5.270000, 5.5900001050, 6.730001050, 6.92100050],
    "weight": [185.23000059, 105.1200099, 110.350003, 190.12000000, 200.59000000]
}

long_decimals_df = pd.DataFrame(long_decimals_info)

"""
Format the data with 2 decimal places
"""
fewer_decimals_df = long_decimals_df.round(decimals=2)
fewer_decimals_df
```

long_to_few_decimals.py hosted with ❤️ by GitHub view raw

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.

```python
import pandas as pd
import numpy as np

candidates_info = {
    'Full_Name':["Aida Kone","Mamadou Diop","Ismael Camara","Aicha Konate",
                 "Fanta Koumare", "Khalil Cisse"],
    'degree':['Master','MS','Bachelor', "PhD", "Masters", np.nan],
    'From':[np.nan,"Dakar","Bamako", "Abidjan","Konakry", "Lomé"],
    'Age':[23,26,19, np.nan,25, np.nan],
          }

candidates_df = pd.DataFrame(candidates_info)

"""
Replace Masters, Master by MS
"""
degrees_to_replace = ["Master", "Masters"]
candidates_df.replace(to_replace = degrees_to_replace, value = "MS", inplace=True)

"""
Replace all the NaN by "Missing"
"""
candidates_df.replace(to_replace=np.nan, value = "Missing", inplace=True)
```

pandas_replace_values.py hosted with ❤️ by GitHub view raw

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.

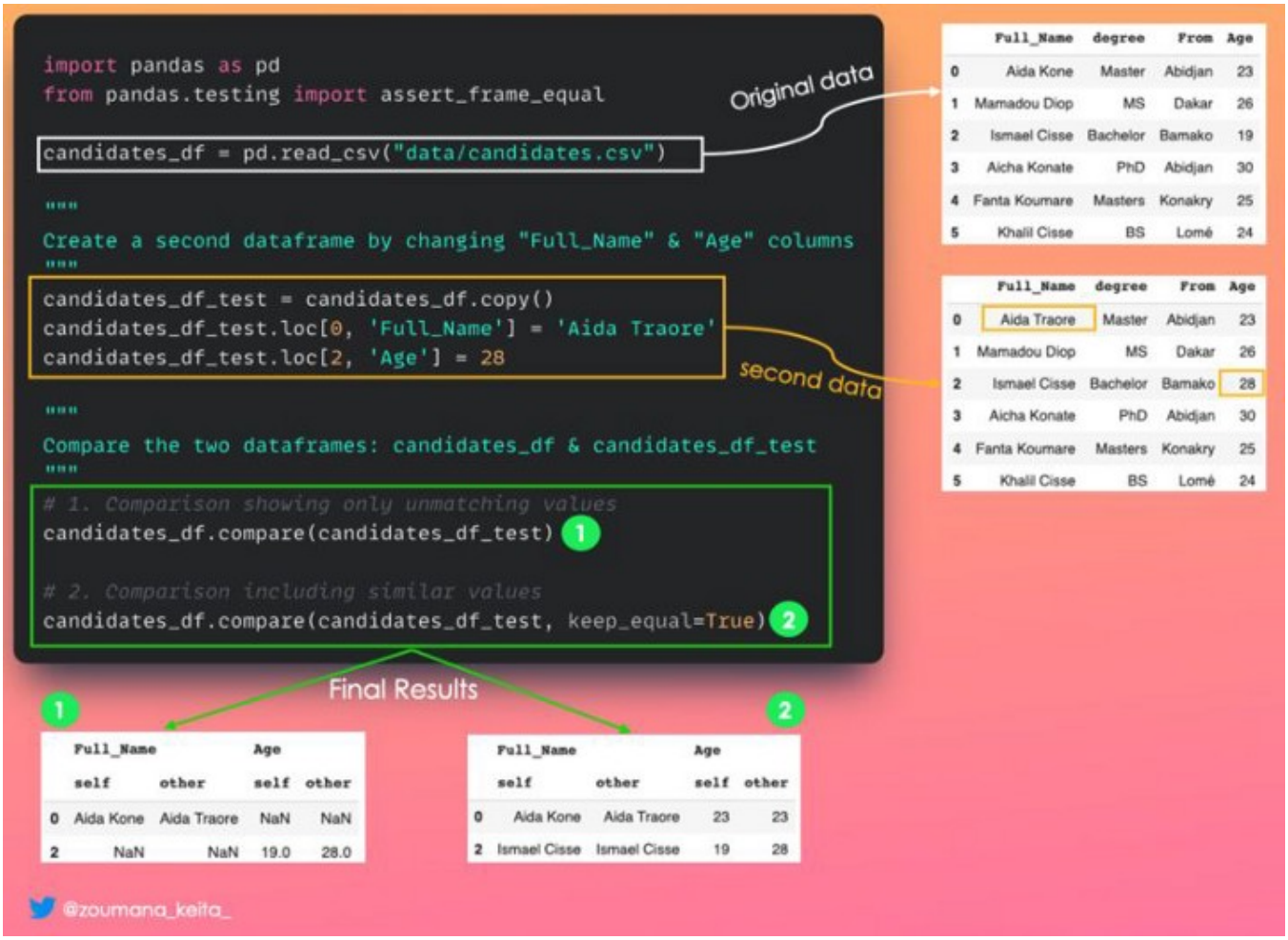
✅ This is where the `.compare()` function comes in handy.

✨ 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`.

```python
import pandas as pd
from pandas.testing import assert_frame_equal

candidates_df = pd.read_csv("data/candidates.csv")

"""
Create a second dataframe by changing "Full_Name" & "Age" columns
"""
candidates_df_test = candidates_df.copy()
candidates_df_test.loc[0, 'Full_Name'] = 'Aida Traore'
candidates_df_test.loc[2, 'Age'] = 28

"""
Compare the two dataframes: candidates_df & candidates_df_test
"""
# 1. Comparison showing only unmatching values
candidates_df.compare(candidates_df_test)

# 2. Comparison including similar values
candidates_df.compare(candidates_df_test, keep_equal=True)
```

pandas_compare_get_differences.py hosted with ❤️ by GitHub view raw



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

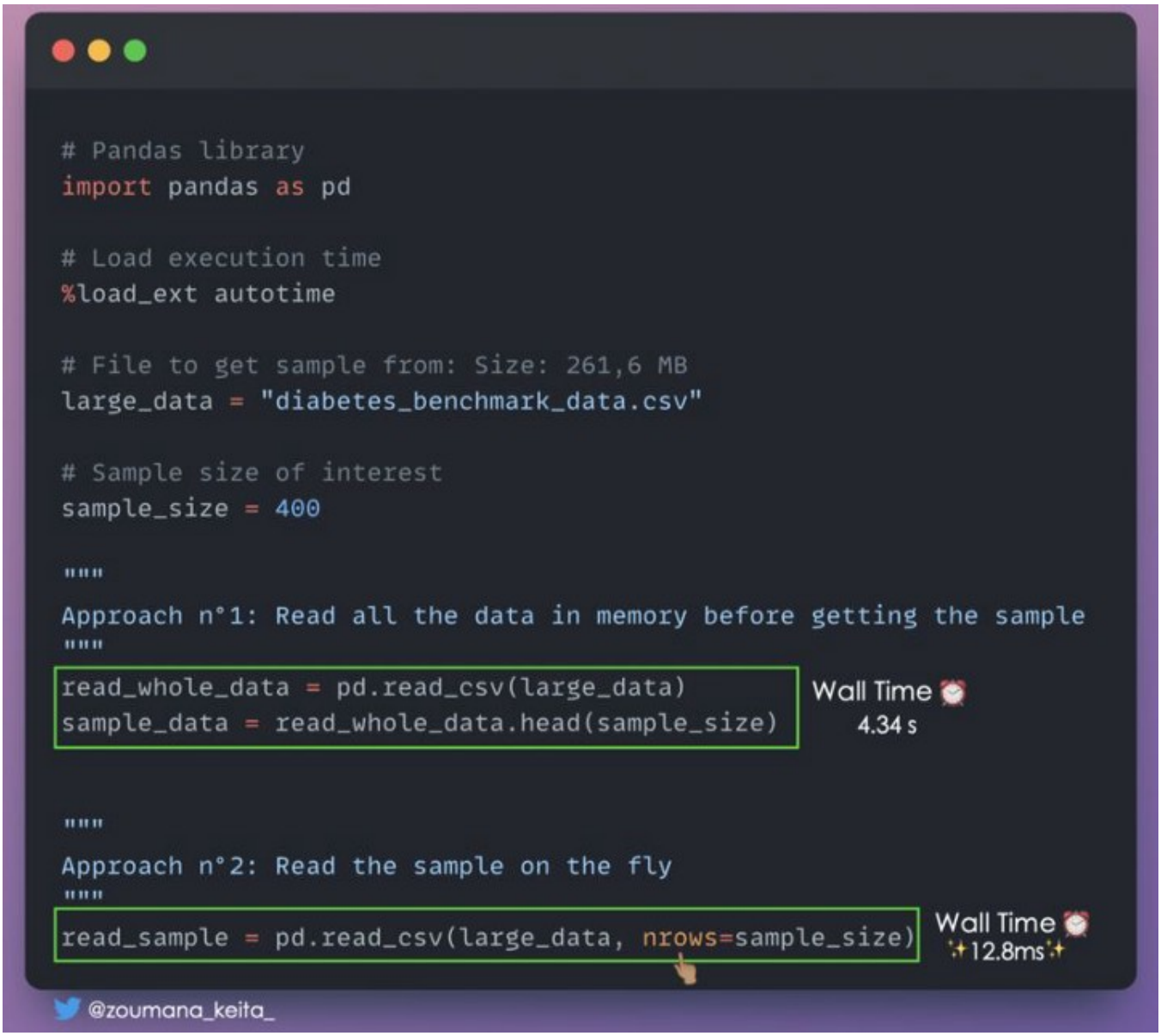
12. Get a subset of a very large dataset for quick analysis

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.

```python
# Pandas library
import pandas as pd

# Load execution time
%load_ext autotime

# File to get sample from: Size: 261,6 MB
large_data = "diabetes_benchmark_data.csv"

# Sample size of interest
sample_size = 400

"""
Approach n°1: Read all the data in memory before getting the sample
"""
read_whole_data = pd.read_csv(large_data)
sample_data = read_whole_data.head(sample_size)

"""
Approach n°2: Read the sample on the fly
"""
read_sample = pd.read_csv(large_data, nrows=sample_size)
```



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.

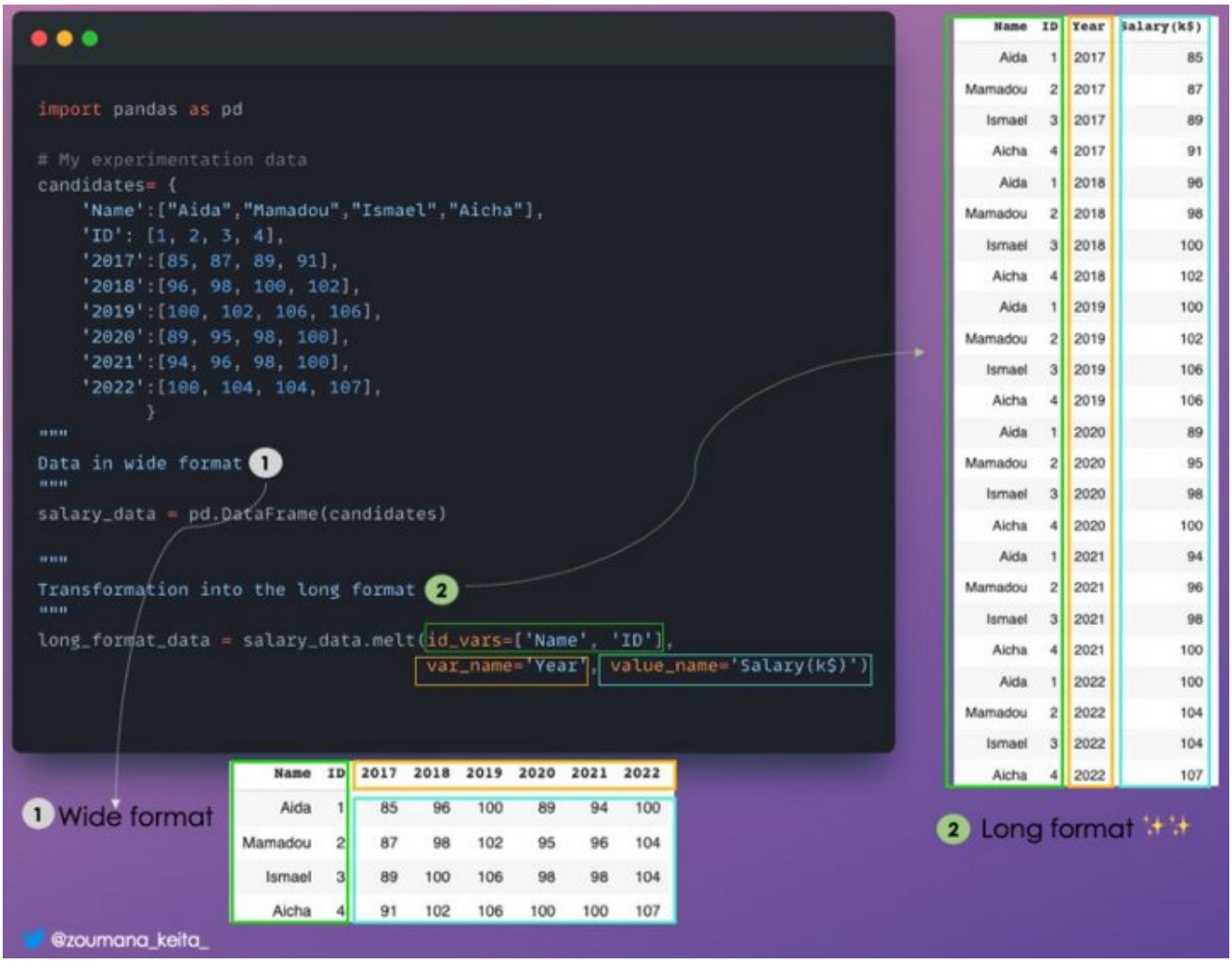
- ***What do you mean by wide*** & ***long***?

✨ 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

```python
import pandas as pd

# My experimentation data
candidates= {
    'Name':["Aida","Mamadou","Ismael","Aicha"],
    'ID': [1, 2, 3, 4],
    '2017':[85, 87, 89, 91],
    '2018':[96, 98, 100, 102],
    '2019':[100, 102, 106, 106],
    '2020':[89, 95, 98, 100],
    '2021':[94, 96, 98, 100],
    '2022':[100, 104, 104, 107],
          }
"""
Data in wide format
"""
salary_data = pd.DataFrame(candidates)

"""
Transformation into the long format
"""
long_format_data = salary_data.melt(id_vars=['Name', 'ID'],
                                    var_name='Year', value_name='Salary(k$)')

```



Name	ID	2017	2018	2019	2020	2021	2022
Aida	1	85	96	100	89	94	100
Mamadou	2	87	98	102	95	96	104
Ismael	3	89	100	106	98	98	104
Aicha	4	91	102	106	100	100	107

Name	ID	Year	Salary(k$)
Aida	1	2017	85
Mamadou	2	2017	87
Ismael	3	2017	89
Aicha	4	2017	91
Aida	1	2018	96
Mamadou	2	2018	98
Ismael	3	2018	100
Aicha	4	2018	102
Aida	1	2019	100
Mamadou	2	2019	102
Ismael	3	2019	106
Aicha	4	2019	106
Aida	1	2020	89
Mamadou	2	2020	95
Ismael	3	2020	98
Aicha	4	2020	100
Aida	1	2021	94
Mamadou	2	2021	96
Ismael	3	2021	98
Aicha	4	2021	100
Aida	1	2022	100
Mamadou	2	2022	104
Ismael	3	2022	104
Aicha	4	2022	107

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

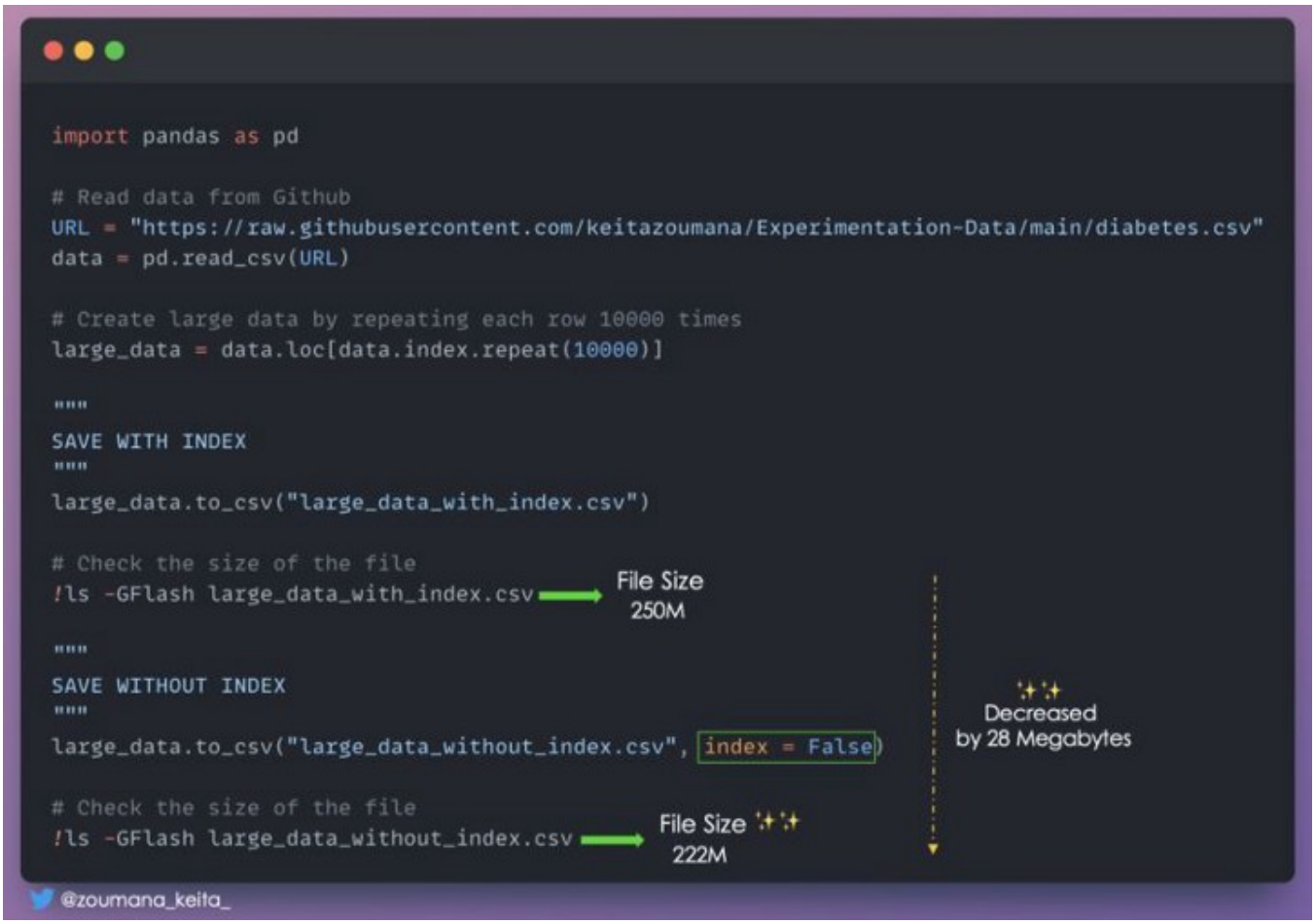
14. Reduce the size of your Pandas data frame by ignoring the index

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.

```python
import pandas as pd

# Read data from Github
URL = "https://raw.githubusercontent.com/keitazoumana/Experimentation-Data/main/diabetes.csv"
data = pd.read_csv(URL)

# Create large data by repeating each row 10000 times
large_data = data.loc[data.index.repeat(10000)]

"""
SAVE WITH INDEX
"""
large_data.to_csv("large_data_with_index.csv")

# Check the size of the file
!ls -GFlash large_data_with_index.csv

"""
SAVE WITHOUT INDEX
"""
large_data.to_csv("large_data_without_index.csv", index = False)

# Check the size of the file
!ls -GFlash large_data_without_index.csv
```



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.

```python
import pandas as pd

# Read data from Github
URL = "https://raw.githubusercontent.com/keitazoumana/Experimentation-Data/main/diabetes.csv"
data = pd.read_csv(URL)

# Create large data for experimentation by repeating each row 20.000 times
exp_data = data.loc[data.index.repeat(20000)]

"""
EXPERIMENT WITH .CSV FORMAT
"""
# Write Time
%%time
exp_data.to_csv("exp_data.csv", index=False)

# Read Time
%%time
csv_data = pd.read_csv("exp_data.csv")

# File Size
!ls -GFlash exp_data.csv

"""
EXPERIMENT WITH .PARQUET FORMAT
"""
# Write Time
%%time
exp_data.to_parquet('exp_data.parquet')

# Read Time
%%time
parquet_data = pd.read_parquet('exp_data.parquet')

# File Size
!ls -GFlash exp_data.parquet
```

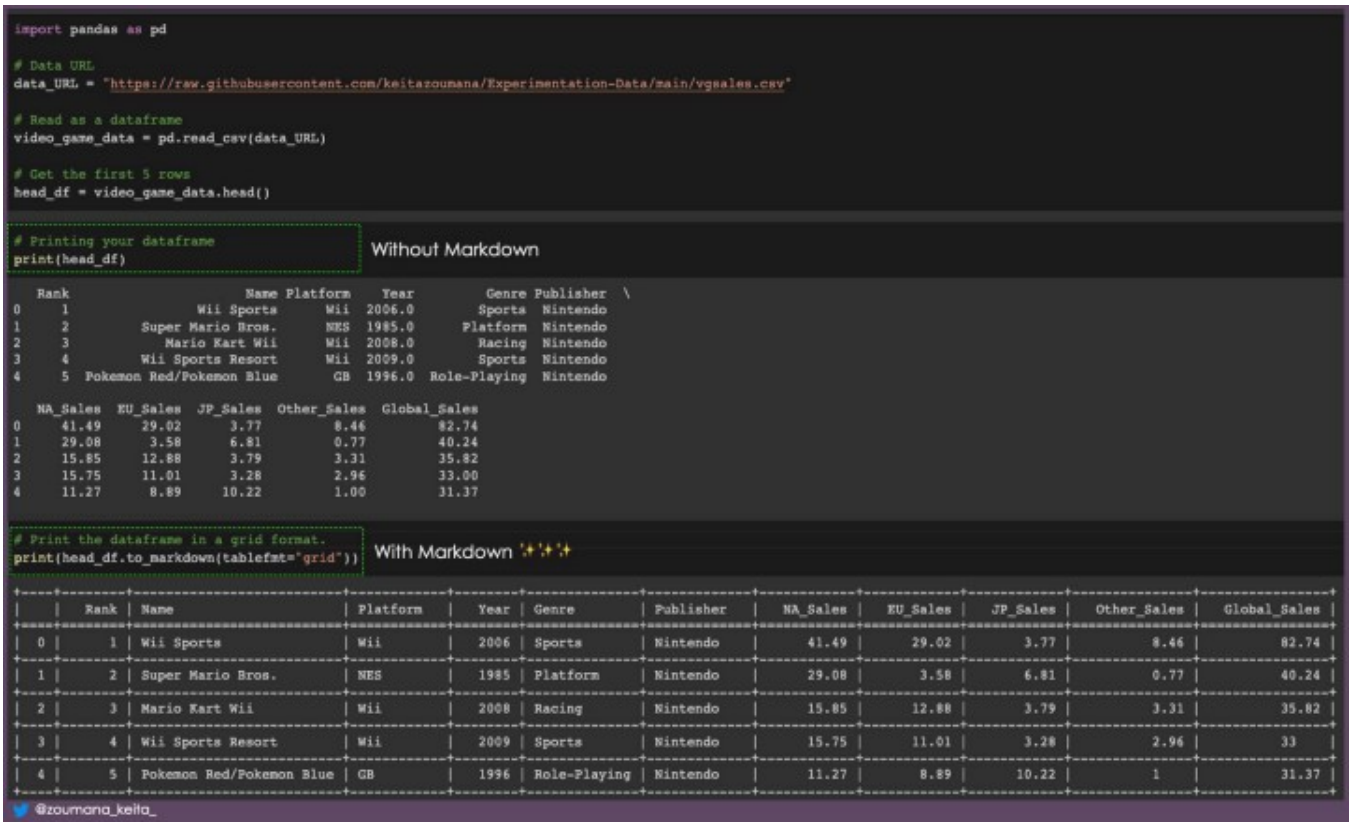
Parquet instead of CSV (Image by Author)

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.

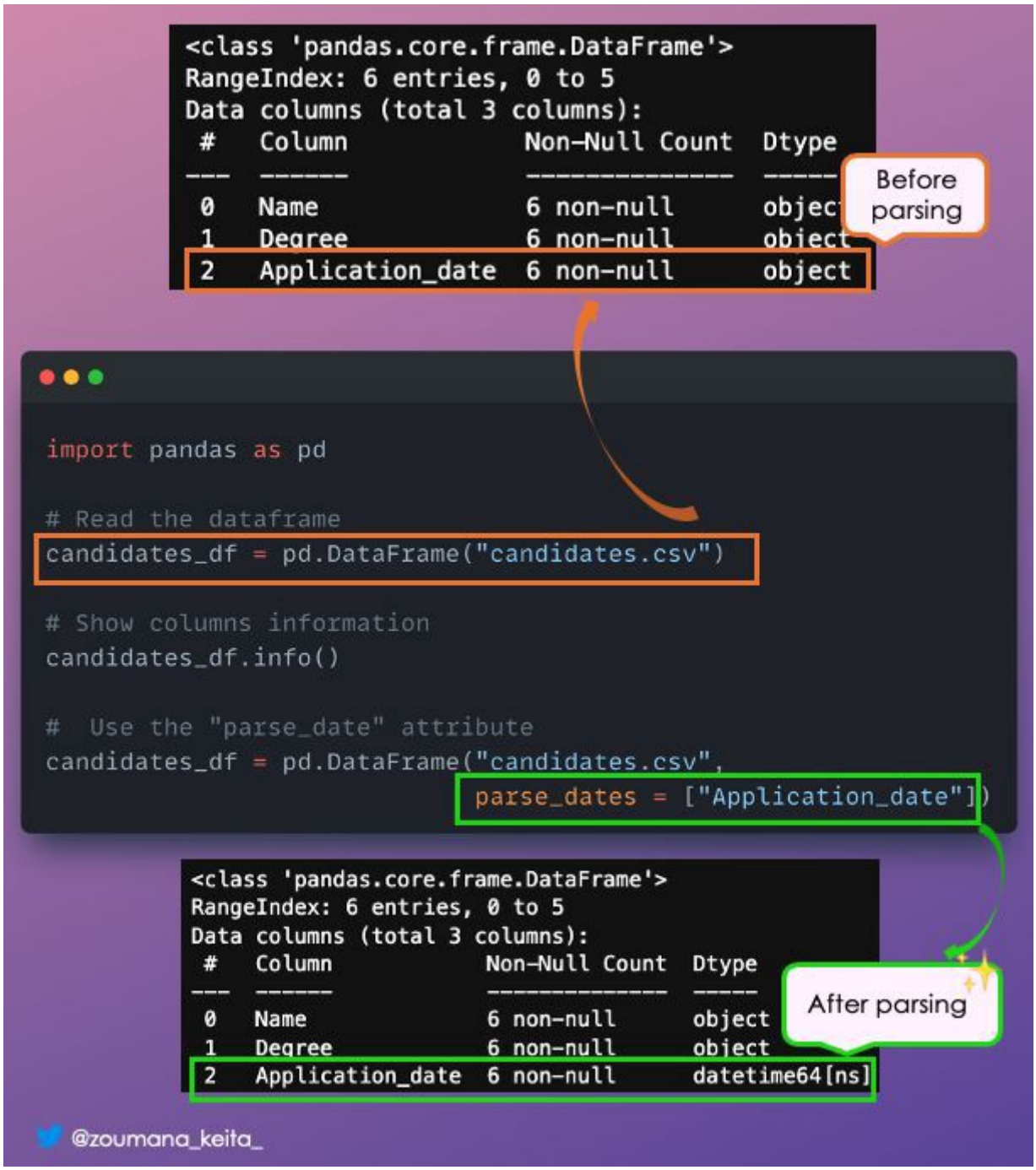
💡 Below is an illustration



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

```
final_dict_doubles = {}

for i in track(range(20), description="Computing 2.n..."):
  final_dict_doubles[f"Value = {i}"] = f"double = {compute_double(i)}"

  # Sleep the process to highligh the progress
  time.sleep(0.8)
```

rich_progress_bar.py hosted with ❤️ by GitHub view raw

rich progress bar implementation

```
for i in tqdm(range(20), desc="Computing 2.n..."):
  final_dict_doubles[f"Value = {i}"] = f"double = {compute_double(i)}"

  # Sleep the process to highligh the progress
  time.sleep(1)
```

tqdm_progress_bar.py hosted with ❤️ by GitHub view raw

tqdm progress bar implementation

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

```python
candidates= {
    'Name':["Aida","Mamadou","Ismael","Aicha","Fatou", "Khalil"],
    'Degree':['Master','Master','Bachelor', "PhD", "Master", "PhD"],
    'From':["Abidjan","Dakar","Bamako", "Abidjan","Konakry", "Lomé"],
    'Application_date': ['11/17/2022', '09/23/2022', '12/2/2021',
                         '08/25/2022', '01/07/2022', '12/26/2022']
          }
candidates_df = pd.DataFrame(candidates)
candidates_df['Application_date'] = pd.to_datetime(candidates_df["Application_date"])

# GET the Values
application_date = candidates_df["Application_date"]

candidates_df["Day"] = application_date.dt.day
candidates_df["Month"] = application_date.dt.month
candidates_df["Year"] = application_date.dt.year
candidates_df["Day_of_week"] = application_date.dt.day_name()
candidates_df["Month_of_year"] = application_date.dt.month_name()
```

use_of_dt_accessor.py hosted with ❤️ by GitHub view raw

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

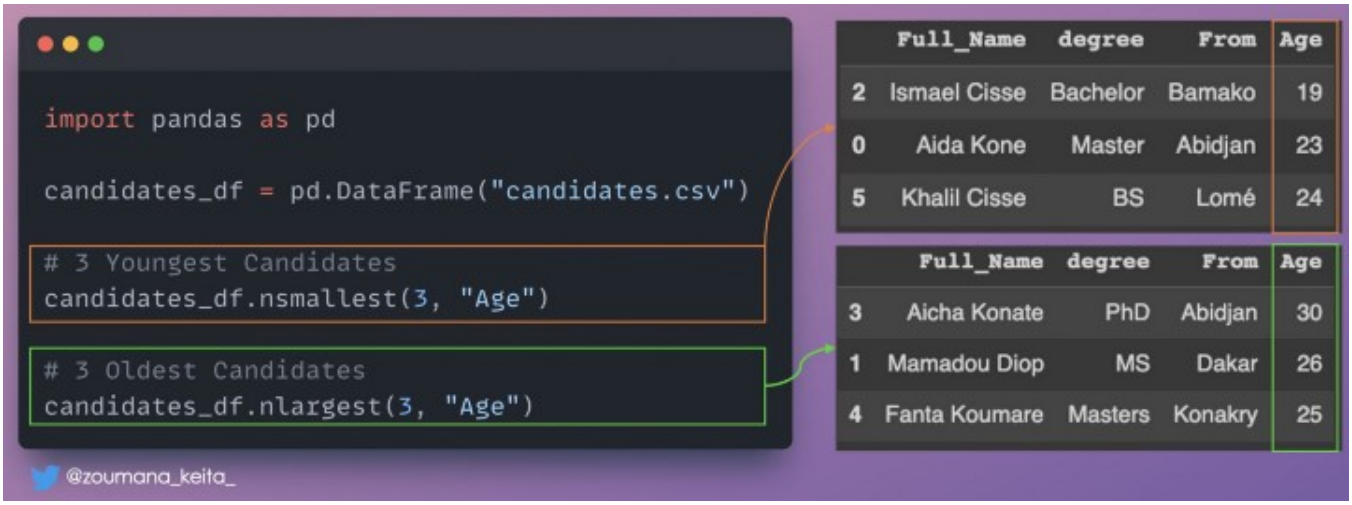
3. Smallest and largest values of a column

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

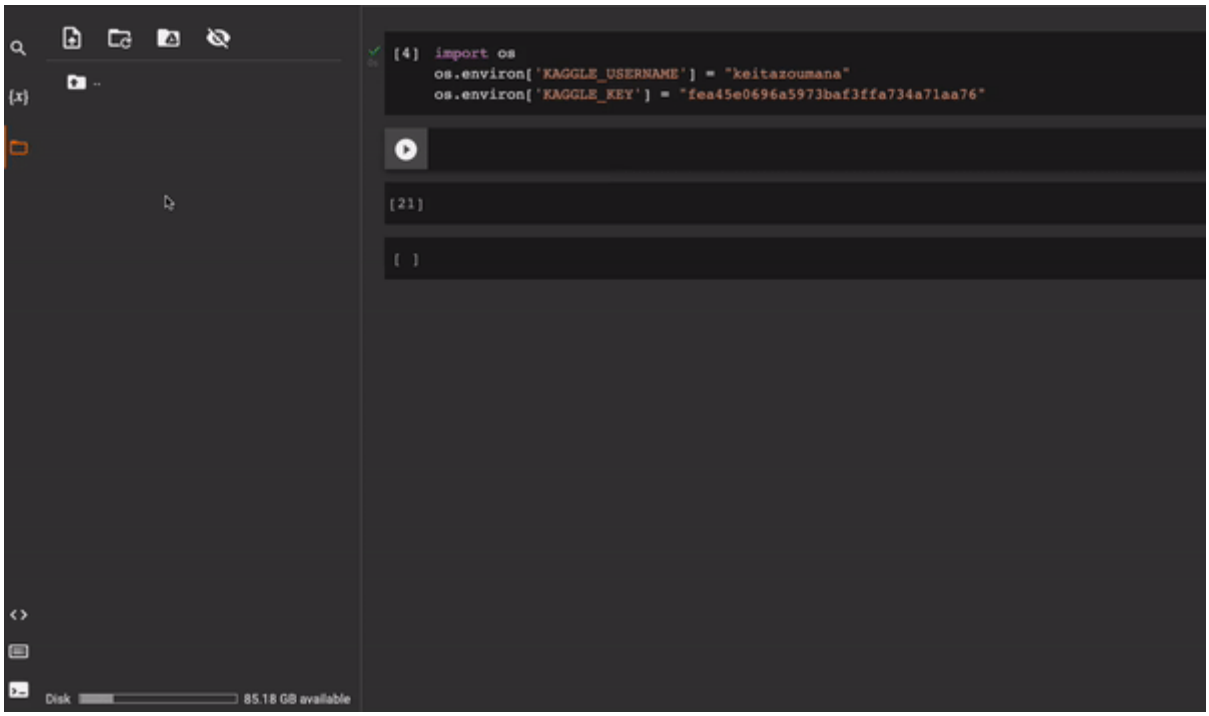


Illustration of %%bash statement (Animation by Autor)

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

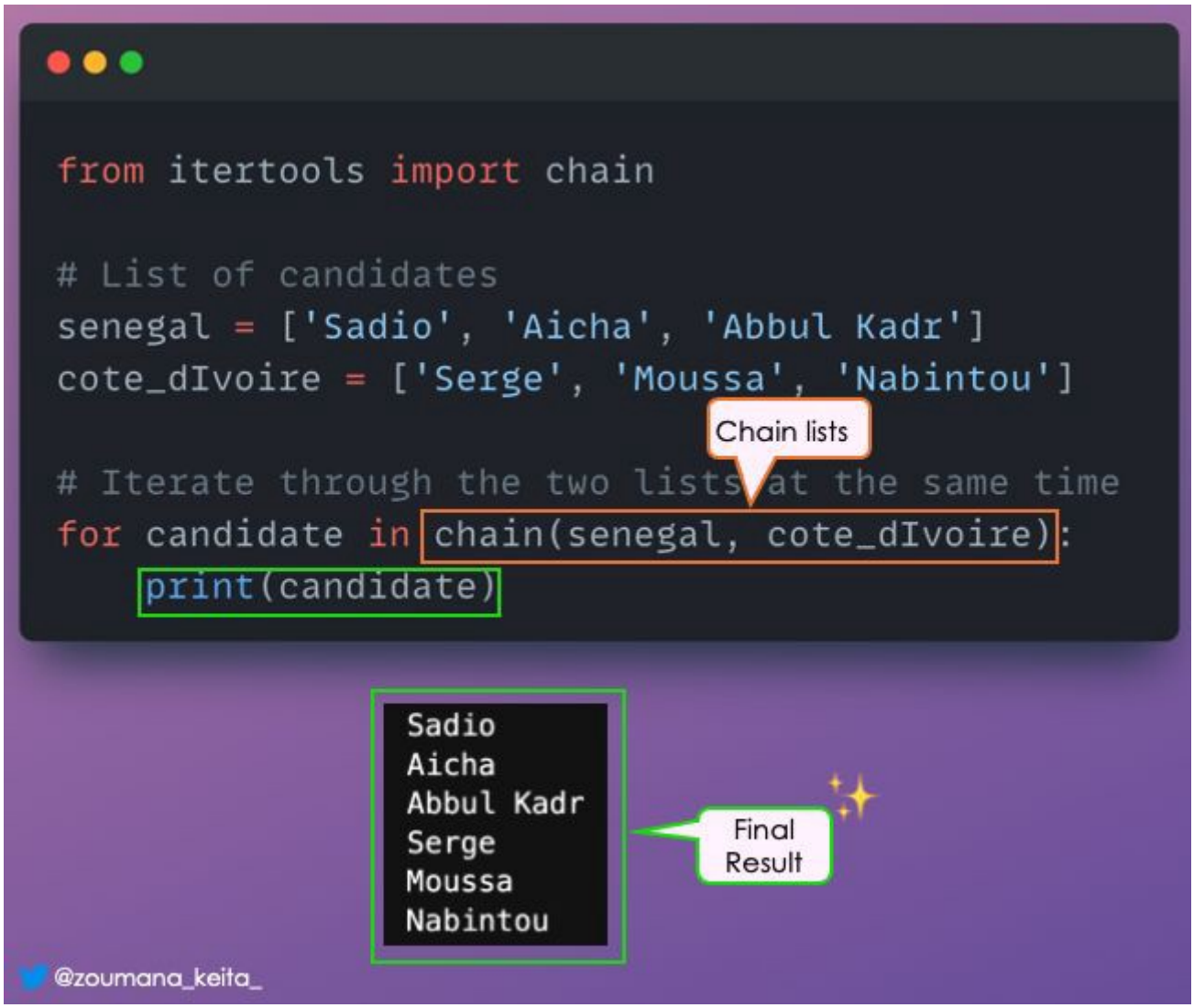
```python
"""
Individual imports
"""
from sklearn.metrics import precision_score, recall_score, f1_score

y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]

print("Precision: ", precision_score(y_true, y_pred, average='macro'))
print("Recall: ", recall_score(y_true, y_pred, average='macro'))
print("F1 Score: ", f1_score(y_true, y_pred, average='macro'))


"""
Single Line import
"""
from sklearn.metrics import precision_recall_fscore_support

precision, recall, f1_score, _ = precision_recall_fscore_support(y_true,
                                                                 y_pred,
                                                                 average='macro')
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1_score}")
```

multiple_metrics.py hosted with ❤️ by GitHub view raw

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.

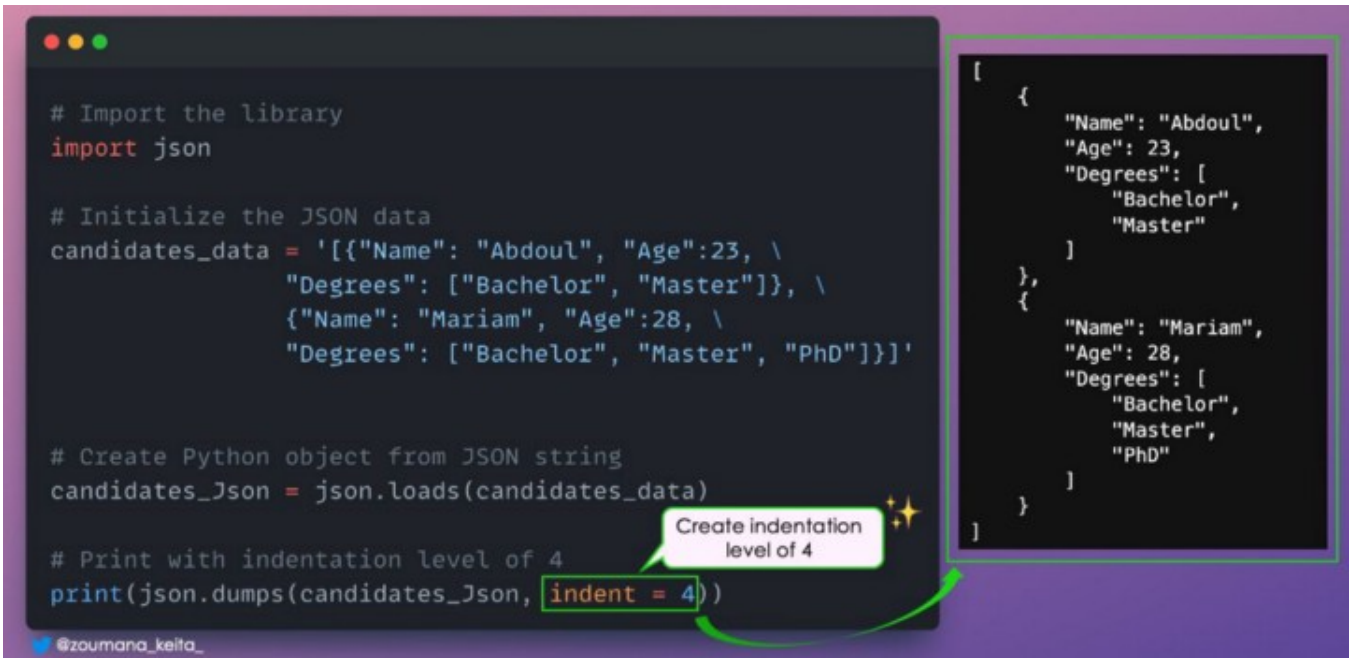


List chaining

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

10. Unit testing

Do you Test Your Code? 🧪

I mean do you perform Unit Testing?

No matter if you are Data Scientist or a Software Developer, Unit testing is an important step to make sure the features being implemented meet the expected behavior.

This is undoubtedly beneficial on many levels:

✨ Better quality 💎 code.

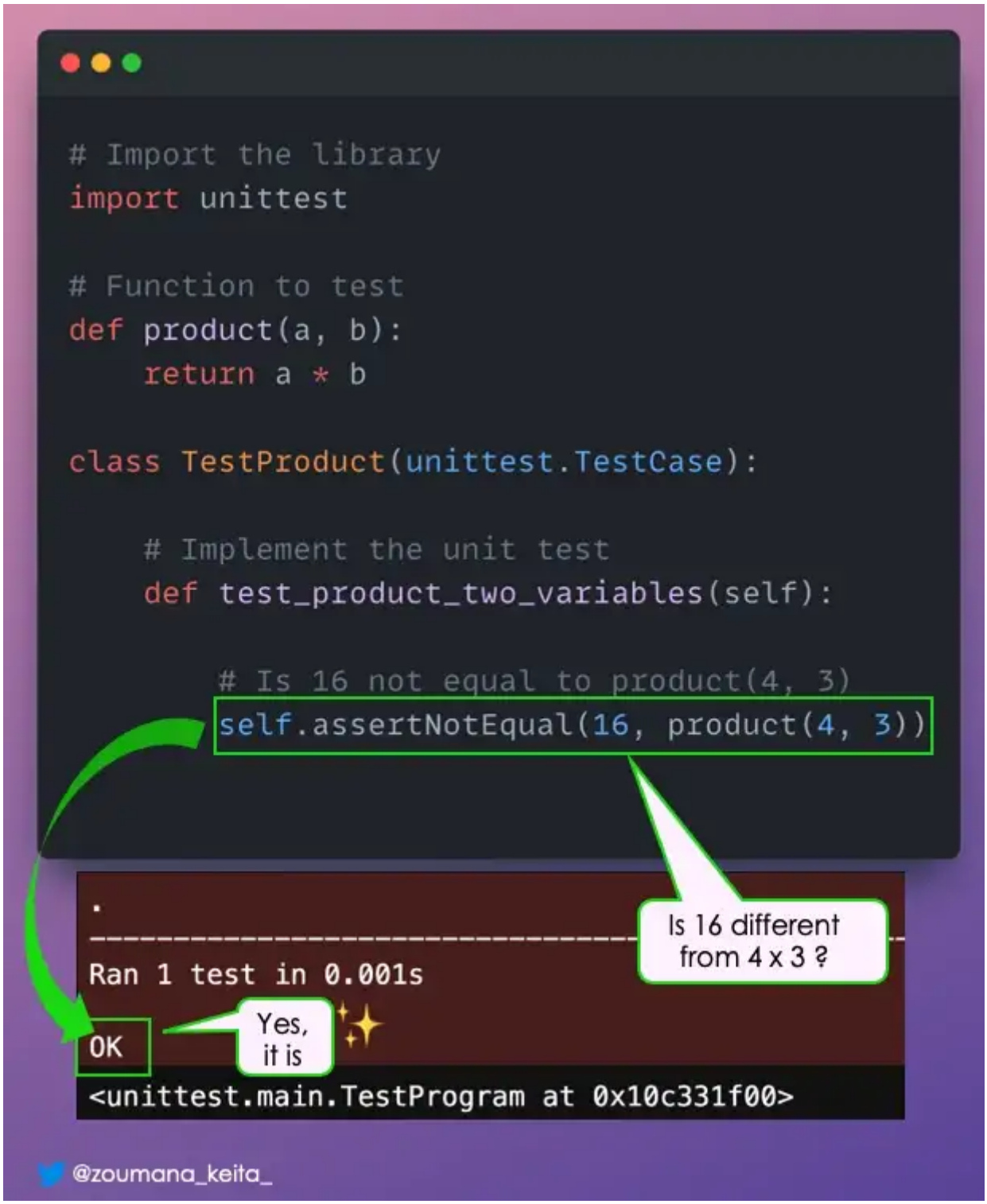
✨ Allows simpler and more agile code when adding new features.

✨ Reduces cost 💰 by saving dev time ⏳ and avoiding later stages of error discovery.

✨ Much More ...

✅ With **unittest**, you can perform unit testing like a pro 😎

Below is an illustration 💡



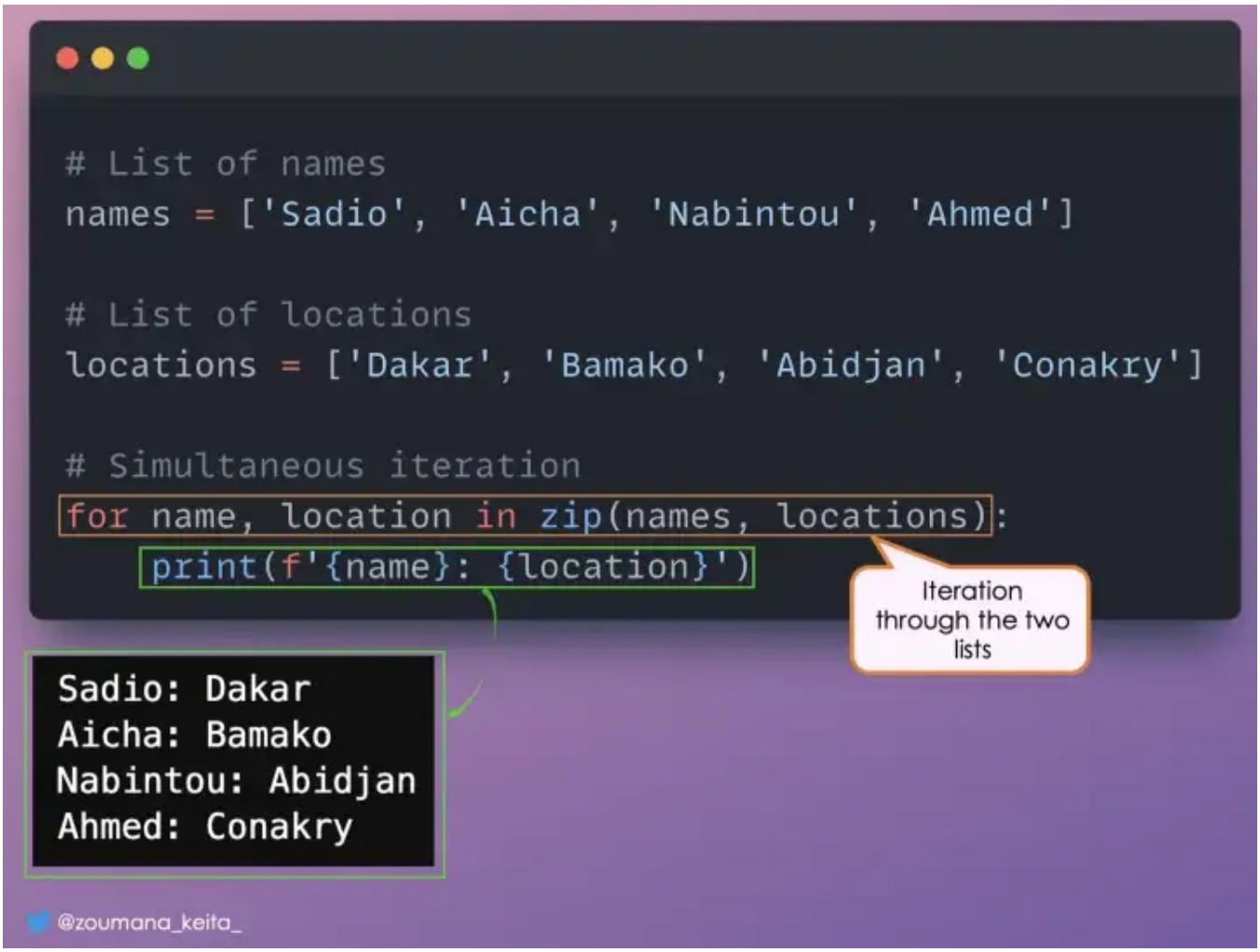
Unit test illustration

11. Iterate over multiple lists

Iterating over multiple lists simultaneously can be beneficial when trying to map ⛓️ information from those lists.

✅ My go-to approach is the Python **zip** function.

Below is an illustration 💡



Iterate over multiple lists

12. Alternative to nested for loops

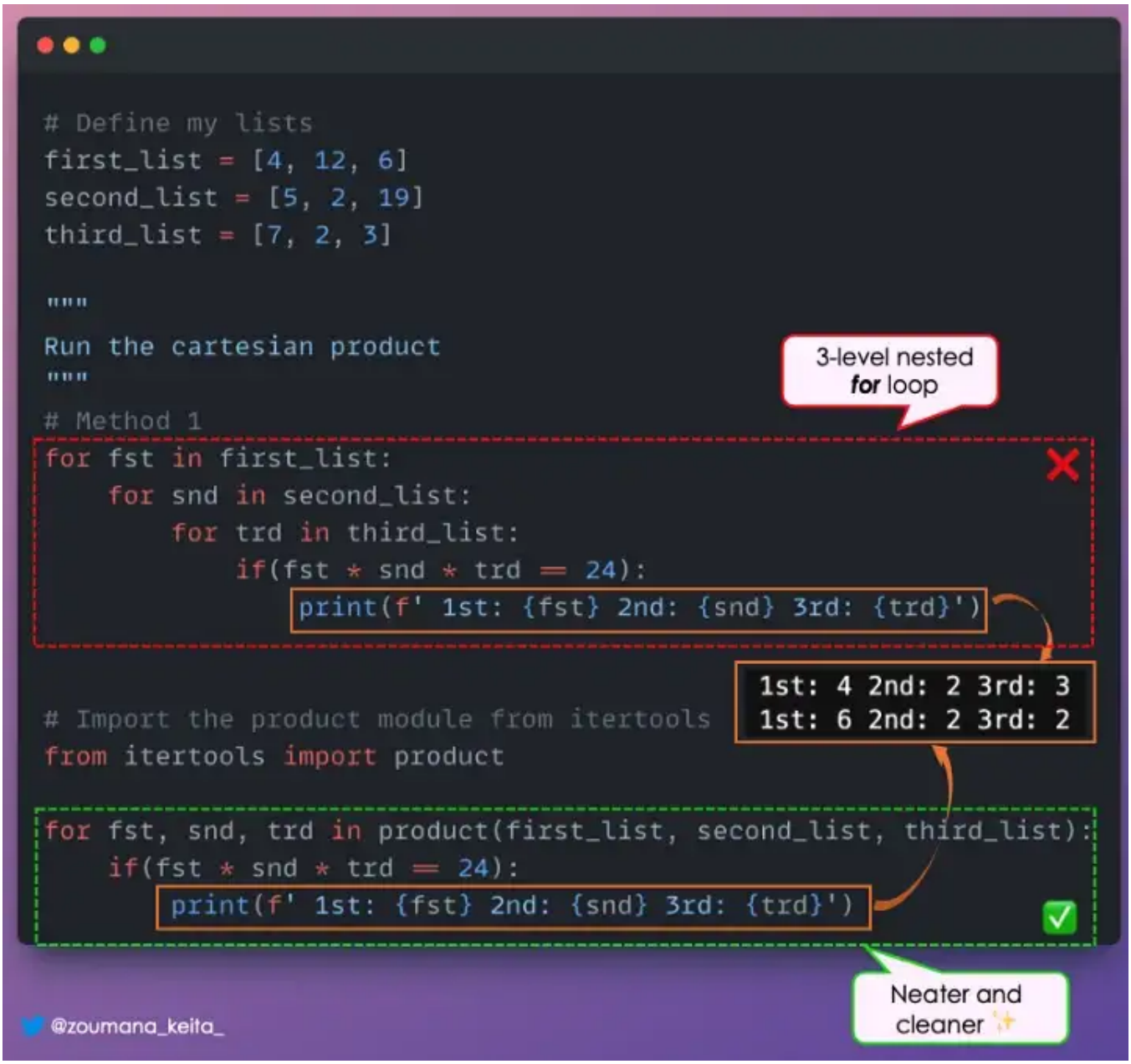
Raise your hand if you have once used nested loops 🔁 🙋🏾

This is most of the time inevitable when a program gets complicated.

However, using nested loops get makes your program harder to read 🔬 and maintain 🤦🏾 .

✅ You can use the Python built-in **product**() function instead.

Below is an illustration 💡



Solution to nested for loops

13. Text preprocessing made easy

Text STX ETX preprocessing has never been easy.

❓ How many functions or regular expressions do you have to write to perform basic text preprocessing tasks like:

✨ Fixing Unicode

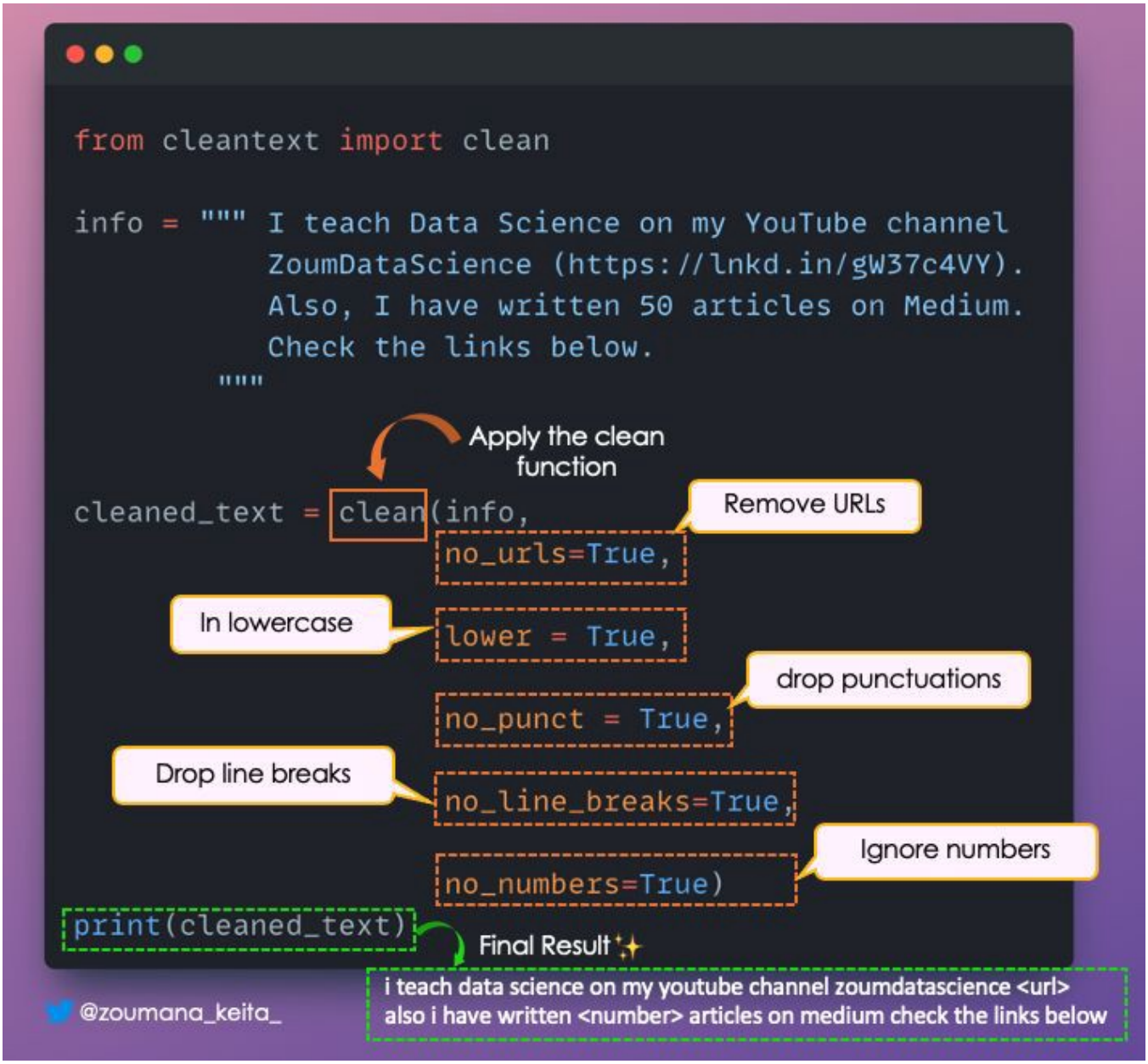
✨ Removing URLs

✨ Getting rid of digits, punctuation, etc?

Those tasks are not only time-consuming ⏰ but also may increase in complexity 📈 depending on the text.

✅ Using the **clean-text** Python library can take away all that burden.

Below is an illustration 💡



Text preprocessing illustration (Image by Author)

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. With a $ 5-a-month commitment, you unlock unlimited access to stories on Medium.