

Communicate

Where are we now? What have we seen/done/studied?

- Data cleaning
- Code organization
 - Moving from notebooks to Python Classes
 - Combining Classes
 - Exploratory Data Analysis
 - Merging tables
 - etc.
- Multivariate Linear Regression
- Logistic Regression

What's next?

- [Cost-Benefit Analysis \(https://www.investopedia.com/terms/c/cost-benefitanalysis.asp\)](https://www.investopedia.com/terms/c/cost-benefitanalysis.asp)
- Summarize your findings
- Communicate

1. Why Communication?

Try to answer this typical interview question for Data Science/Analytics jobs:

"Tell me about a project where you *actually* changed the course of action of your business because of your data."

Here, proper communication is **very** important!

Let's look at how we can:

- Influence! 🖱️ **Data Science as U-Turn** (= using data to influence directions)
- Collaborate
- Write well-documented code to ensure it can be replicated

2. Qualities of a Well-Communicated DS Project

Make the complex easy to understand

What is well understood is clearly said, And the words to say it flow with ease.

Original: Ce que l'on conçoit bien, s'énonce clairement, Et les mots pour le dire arrivent aisément.

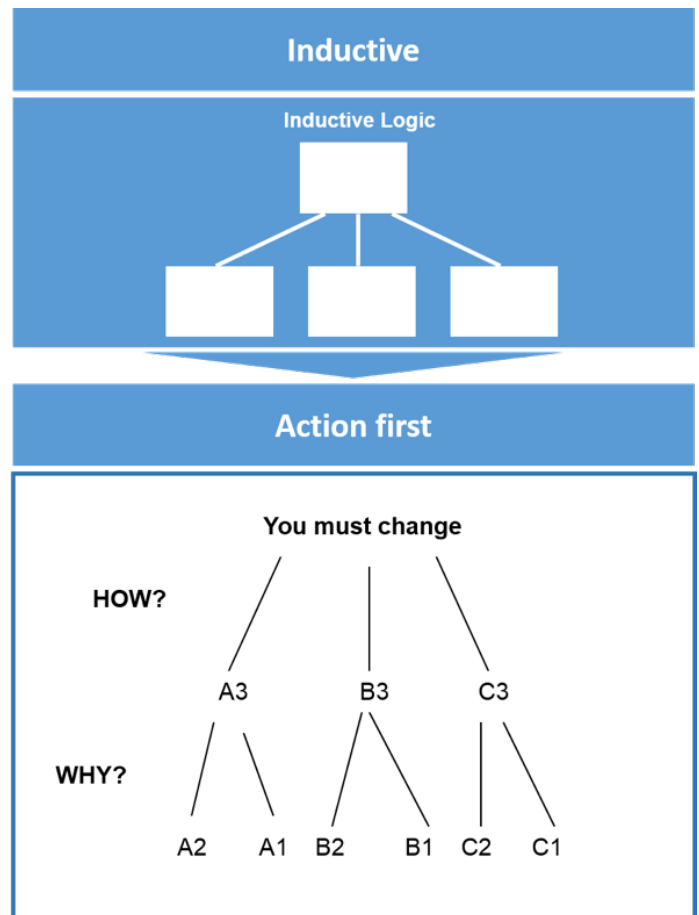
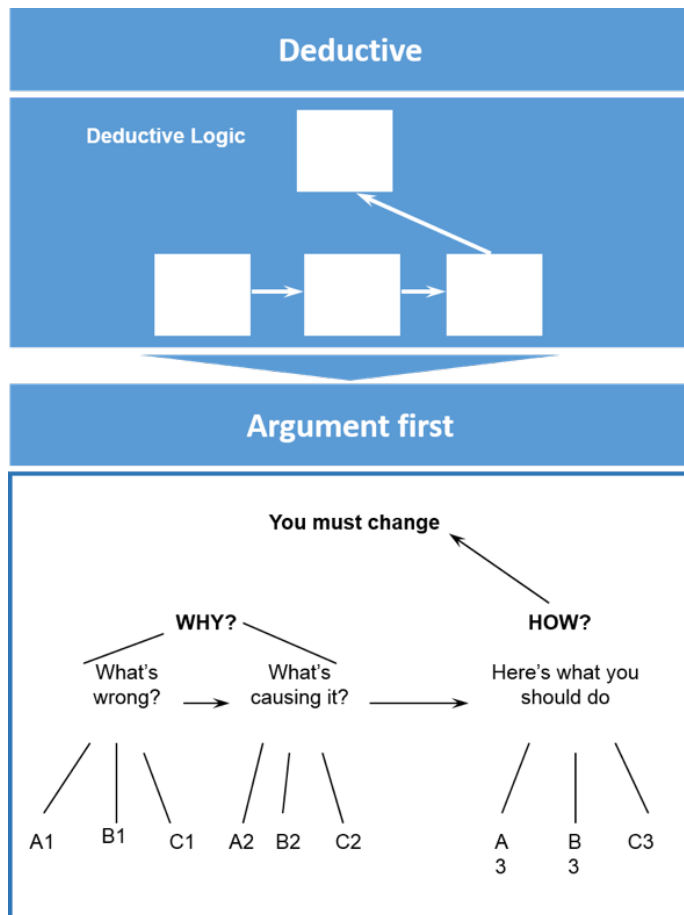
🇫🇷 [Nicolas Boileau-Despréaux](https://en.wikipedia.org/wiki/Nicolas_Boileau-Despr%C3%A9aux) (https://en.wikipedia.org/wiki/Nicolas_Boileau-Despr%C3%A9aux), L'Art poétique (1674)

⚡ **Be Concise!**

Follow the [Pyramid Principle](https://gettingbettereveryday.org/2018/10/05/what-you-could-learn-from-barbra-mintos-the-pyramid-principle-2009-172-pages/) (<https://gettingbettereveryday.org/2018/10/05/what-you-could-learn-from-barbra-mintos-the-pyramid-principle-2009-172-pages/>)

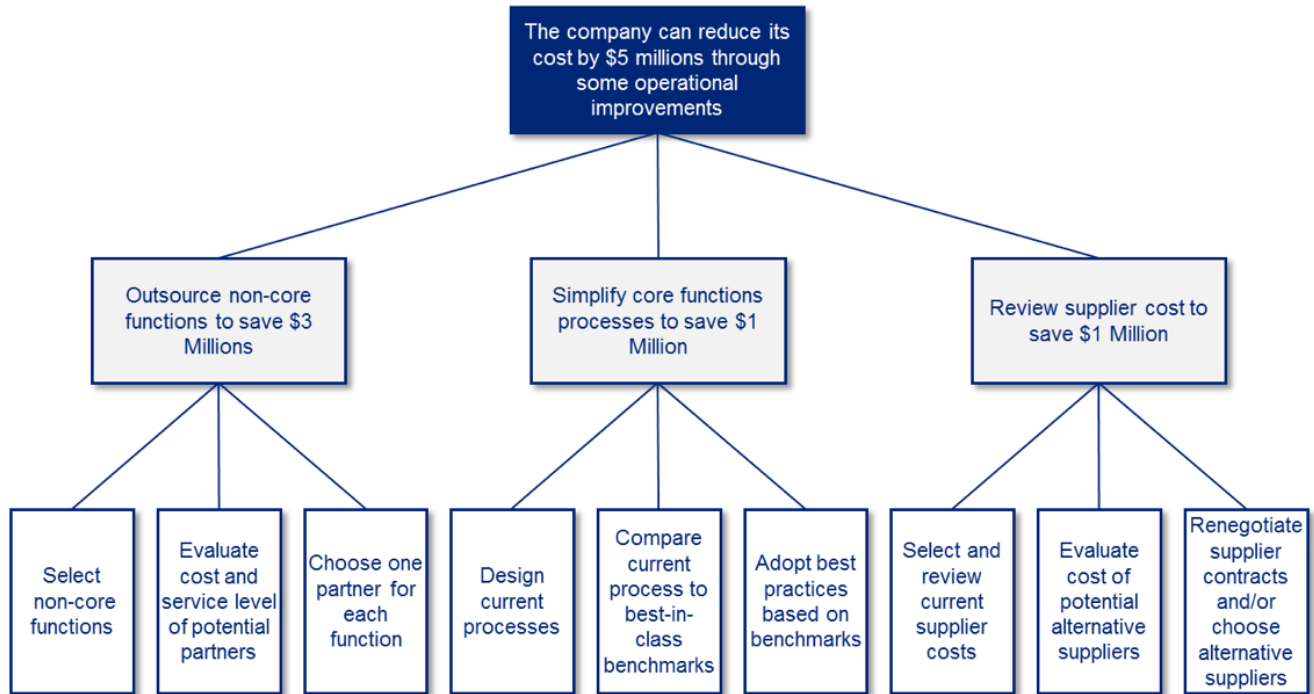
👉 Action/Answer first, explain why after!

👉 More efficient under time constraints

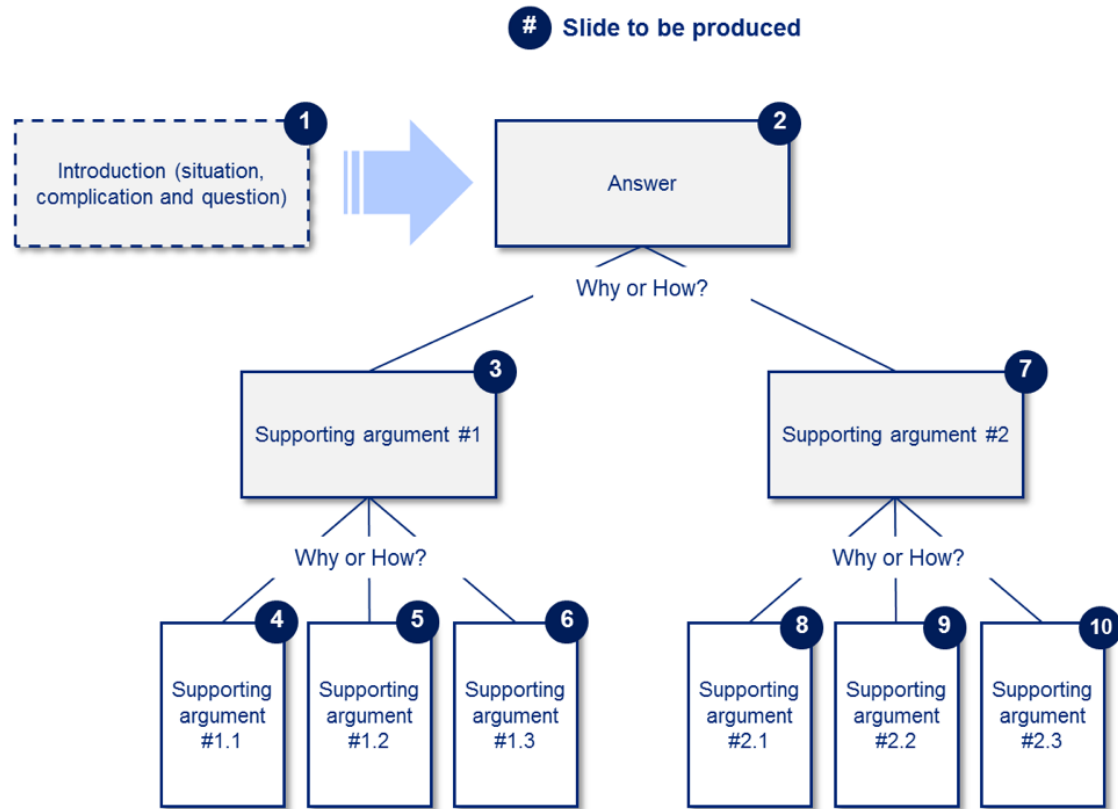


Create a pyramid with your answer and supporting arguments - Example

How can the company reduce its cost by 10%?



Based on your pyramid, identify the slides that you need to produce



3. Understand Trade-Offs

Consider the following example

Your **Product Manager (PM)** comes up with an idea to **increase the conversion rate** on a website

- This website has ~100K visitors per month
- On average, a visit provides a ~20 BRL benefit (100K * 20 BRL = 2.000.000 BRL)
- You have a team of 3 engineers who cost 500 BRL/day each

👉 You don't know **how long** it will take to code this feature

👉 You don't know the **lift** (increase) this potential feature could have on the conversion rate.

? How do you inform your product manager ?

Step 1 - Model Benefits

We estimate the potential lifts between +1% and +4%

```
In [ ]: lift = np.arange(0.01, 0.05, 0.01)

benefits = pd.DataFrame(
    {'lift': lift, 'benefit': 100000 * 20 * lift}
)

benefits
```

Out[]:

	lift	benefit
0	0.01	20000.0
1	0.02	40000.0
2	0.03	60000.0
3	0.04	80000.0

Step 2 - Model Costs

- f
(number of workdays of engineering team)
- Estimate between 20 and 50 days

```
In [ ]: workdays = np.arange(20, 60, 10)

costs = pd.DataFrame(
    {'workdays': workdays, 'cost': workdays * 500 * 3}
)

costs
```

Out[]:

	workdays	cost
0	20	30000
1	30	45000
2	40	60000
3	50	75000

Step 3 - Sensitivity Matrix

We can highlight results in a **sensitivity matrix** to see the **trade-off**:

workdays	20	30	40	50
lift				
0.01	-10000.0	-25000.0	-40000.0	-55000.0
0.02	10000.0	-5000.0	-20000.0	-35000.0
0.03	30000.0	15000.0	0.0	-15000.0
0.04	50000.0	35000.0	20000.0	5000.0

workdays	cost	lift	benefit
0	20 30000	0 0.01	20000.0
1	30 45000	1 0.02	40000.0
2	40 60000	2 0.03	60000.0
3	50 75000	3 0.04	80000.0

Step 3.1

We have to create the **cartesian product** of both the cost and the benefit DataFrame s

```
In [ ]: sensitivity = costs.merge(benefits, how='cross')
sensitivity
```

Out[]:

	workdays	cost	lift	benefit
0	20	30000	0.01	20000.0
1	20	30000	0.02	40000.0
2	20	30000	0.03	60000.0
3	20	30000	0.04	80000.0
4	30	45000	0.01	20000.0
5	30	45000	0.02	40000.0
6	30	45000	0.03	60000.0
7	30	45000	0.04	80000.0
8	40	60000	0.01	20000.0
9	40	60000	0.02	40000.0
10	40	60000	0.03	60000.0
11	40	60000	0.04	80000.0
12	50	75000	0.01	20000.0
13	50	75000	0.02	40000.0
14	50	75000	0.03	60000.0
15	50	75000	0.04	80000.0

Step 3.2

We can now add `net_profit` and clean up the sensitivity matrix


```
In [ ]: sensitivity['net_profit'] = sensitivity['benefit'] - sensitivity['cost']
sensitivity.drop(columns=['cost', 'benefit'], inplace=True)  # Dropping columns we don't need
sensitivity.head()
```

Out[]:

	workdays	lift	net_profit
0	20	0.01	-10000.0
1	20	0.02	10000.0
2	20	0.03	30000.0
3	20	0.04	50000.0
4	30	0.01	-25000.0

```
In [ ]: # Pivot is the easiest way to do it
sensitivity.pivot(index='lift', columns='workdays', values='net_profit')
```

Out[]:

	workdays	20	30	40	50
lift					
0.01		-10000.0	-25000.0	-40000.0	-55000.0
0.02		10000.0	-5000.0	-20000.0	-35000.0
0.03		30000.0	15000.0	0.0	-15000.0
0.04		50000.0	35000.0	20000.0	5000.0

```
In [ ]: # Creating a double index and unstacking also works!
sensitivity.set_index(['lift', 'workdays']).unstack()
```

Out[]:

		net_profit			
	workdays	20	30	40	50
lift					
0.01		-10000.0	-25000.0	-40000.0	-55000.0
0.02		10000.0	-5000.0	-20000.0	-35000.0
0.03		30000.0	15000.0	0.0	-15000.0
0.04		50000.0	35000.0	20000.0	5000.0

```
In [ ]: # The underscore used here is a useful way to grab the last output
        _.style.map(lambda x : 'color: red' if x < 0 else 'color: black')
```

Out[]:

	net_profit			
workdays	20	30	40	50
lift				
0.010000	-10000.000000	-25000.000000	-40000.000000	-55000.000000
0.020000	10000.000000	-5000.000000	-20000.000000	-35000.000000
0.030000	30000.000000	15000.000000	0.000000	-15000.000000
0.040000	50000.000000	35000.000000	20000.000000	5000.000000

4. Powerful Forms of Communication

1. Docs and code itself (see [Communication Through Code \(https://medium.com/the-mighty-programmer/code-communication-ac4a8a4a4f9a\)](https://medium.com/the-mighty-programmer/code-communication-ac4a8a4a4f9a))
2. Interactive tools
3. Notebooks

4.1 - Code and Docs

Write the Docs



- [Write the Docs \(https://www.writethedocs.org/\)](https://www.writethedocs.org/) is a community dedicated to the art and science of technical documentation
- Why good technical documentation?
 - Your code in 6 months will likely belong to someone else
 - Well-documented code helps others understand what it does
 - Detailing your methodology makes it easy for others to follow it
 - If you come back to undocumented code, you have no idea what you did 🤔

Even just a week after writing it! 🤔

args and kwargs

```
In [ ]: def foo(required, *args, **kwargs):  
        """This is the function docstring"""  
  
        print(required)  
  
        if args:  
            print(args)  
        if kwargs:  
            print(kwargs)
```

```
In [ ]: foo(1, 2, 3, bar = 5)  
  
1  
(2, 3)  
{'bar': 5}
```

```
In [ ]: a_list = [1, 2, 3]
        a_dict = {'bar': 5}

        foo(*a_list, **a_dict)
```

```
1
(2, 3)
{'bar': 5}
```

```
In [ ]: class Student:
        school = 'lewagon'

        def __init__(self, name, age):
            self.name = name
            self.age = age
```

```
In [ ]: alice = Student('alice', 20)
        alice.__dict__
```

```
Out[ ]: {'name': 'alice', 'age': 20}
```

```
In [ ]: d = {'a': 1, 'b': 2}
        d.update(c = 3)

        d
```

```
Out[ ]: {'a': 1, 'b': 2, 'c': 3}
```

```
In [ ]: # Student class with allowed optional attributes
        class Student:
            school = 'lewagon'

            def __init__(self, name, age, **kwargs):
                self.name = name
                self.age = age

                self.__dict__.update(**kwargs)
```

```
In [ ]: alice = Student('alice', 20, nationality='brazilian')
        alice.nationality
```

```
Out[ ]: 'brazilian'
```

```
In [ ]: alice.__dict__
```

```
Out[ ]: {'name': 'alice', 'age': 20, 'nationality': 'brazilian'}
```

```
In [ ]: # DataStudent child class with abstract arguments
class DataStudent(Student):
    course = 'data'

    # CAREFUL: the batch parameter should
    # not be passed to the parent class!
    def __init__(self, name, age, batch, **kwargs):
        super().__init__(name, age, **kwargs)

        self.batch = batch          # now we can add a batch!
```

```
In [ ]: DataStudent('alice', 20, '#1207', nationality='brazilian').__dict__
```

```
Out[ ]: {'name': 'alice', 'age': 20, 'nationality': 'brazilian', 'batch': '#1207'}
```

Typing Hints

Recent versions of Python allow you to explicitly specify the **datatype** needed for both the function's inputs and outputs

[Real Python - Type Checking \(https://realpython.com/python-type-checking/\)](https://realpython.com/python-type-checking/)

```
In [ ]: def say_hi(name: str) -> str:
        return name + ' says Hi!'

say_hi('alice')
```

```
Out[ ]: 'alice says Hi!'
```

```
In [ ]: def cost_of_reviews(s: pd.Series) -> int:
        return s.map({
            1: 100,
            2: 50,
            3: 40,
            4: 0,
            5: 0
        }).sum()

cost_of_reviews(pd.Series([1, 1, 5, 4]))
```

```
Out[ ]: 200
```

4.2 - Interactive Tools

```
In [ ]: # start at 6min30, stops at 9min17
        from IPython.display import HTML, IFrame

        IFrame("http://www.youtube.com/embed/8QiPFmIMxFc?t=388", width="560",
                height="315")
```

Out[]:



[Bret Victor's "Inventing on Principle" \(http://www.youtube.com/embed/8QiPFmIMxFc?t=388\)](http://www.youtube.com/embed/8QiPFmIMxFc?t=388), on the need for interactivity and fast feedback loops in creative processes

ipywidgets

```
from ipywidgets import interact

@interact
def plot_polynom(a=[0, 1, 2, 3], b=2):
    x = np.arange(-10, 10, 0.1)
    y = a*x**3+ b*x**2

    plt.plot(x,y); plt.xlim(xmin=-10, xmax=10); plt.ylim(ymin=-100, ymax=100)
```

Python decorators @

```
In [ ]: def my_decorator(func):
        def wrapper():
            print("I'm before the method call")
            func()
            print("I'm after the method call")

            return wrapper

        def say():
            print("Hi!")

        say = my_decorator(say)

        say()
```

```
I'm before the method call
Hi!
I'm after the method call
```

```
In [ ]: def my_decorator(func):  
        def wrapper():  
            print("I'm before the method call")  
            func()  
            print("I'm after the method call")  
  
        return wrapper  
  
@my_decorator  
def say():  
    print("Hi!")  
  
say()
```

```
I'm before the method call  
Hi!  
I'm after the method call
```

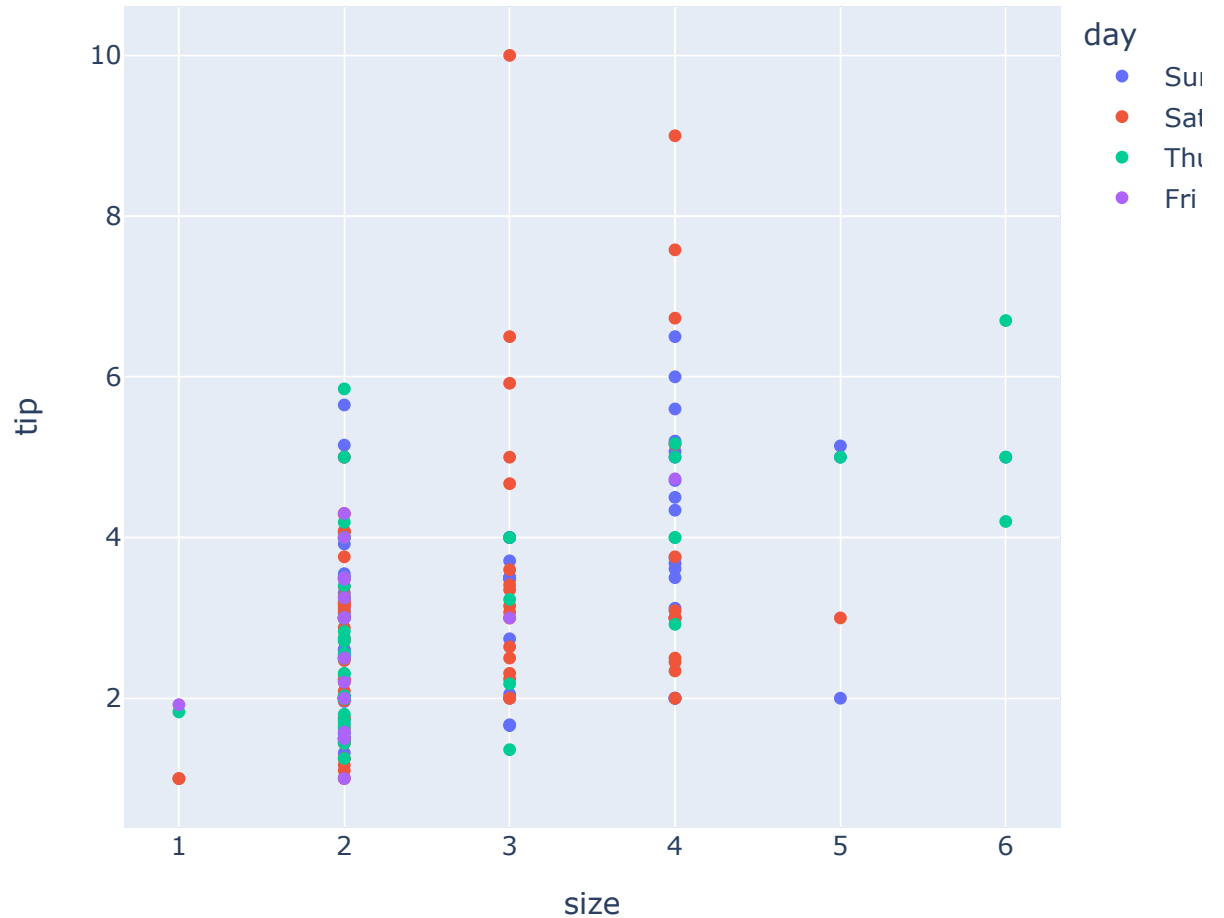
@my_decorator is just an easier way of **wrapping** features around existing methods

- Easy to add (does not depend on your function name)
- Easy to remove (your function name does not stay modified after removal)

 For the curious among you, here is a [great blog post \(https://realpython.com/primer-on-python-decorators/\)](https://realpython.com/primer-on-python-decorators/) on Python decorators (1h read)!

Plotly Express


```
df = px.data.tips()
df
fig = px.scatter(df, x="size", y="tip", color="day")
fig.show()
```



4.3 - Notebook-Based Presentations

Step 1: Prepare a Clean Notebook

- Enable selection of cell slide type
 - **Jupyter Notebook:** View → Right Sidebar → Show Notebook Tools → Common Tools
 - **Jupyter Lab:** View → Appearance → Show Right Sidebar
 - **VS Code:** Right-Click on a cell's status bar (bottom) → Switch Slide Type
- Select the slide type for each cell
 - Slide → main chapters
 - Sub-Slide → new slide below the main chapter
 - Fragment → subset of the slide directly above, appears when pressing the arrow down
 - - = display immediately after the cell above
 - Skip = does not display (e.g. hides code logic)
 - Notes = does not display, for private notes

Step 2: Convert to a Static Slide-Based HTML Doc with Jupyter nbconvert

```
jupyter nbconvert --to slides --post serve <your_notebook.ipynb>
```

Some Additional Tips

- Use **Markdown** for...everything! 🖱️ [Markdown cheatsheet \(https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet\)](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet)
- Use **HTML** for style and images 🖱️ [HTML cheatsheet \(https://htmlcheatsheet.com/\)](https://htmlcheatsheet.com/)
- Use *LaTeX* for math! 🖱️ [LaTeX cheatsheet \(https://wch.github.io/latexsheet/latexsheet-a4.pdf\)](https://wch.github.io/latexsheet/latexsheet-a4.pdf)

🖱️ Use *emojis* (macOS: ⌘-^–Space , Windows: Win-.)

- Esc - M changes a cell to Markdown mode
- Esc - Y changes it back to code mode

Remember: if you want to share an interactive notebook, be careful with dependencies (imports/packages)


Notebook Best Practices

- Use Collapsible Headings and Table of Content
- Notebooks should be executable from top to bottom
- Name your variables carefully
- Use dummy names such as `tmp` or `_` when needed
- Clear useless variables from RAM when not needed (`del my_variable`)
- Clear your code and merge cells when relevant (`Shift-M`)
- Hide your cell outputs to gain space (double-click on the red `Out[]:` section to the left of your cell).

Bibliography

-  [Thinking Fast and Slow \(https://en.wikipedia.org/wiki/Thinking,_Fast_and_Slow\)](https://en.wikipedia.org/wiki/Thinking,_Fast_and_Slow)
-  [The Pyramid Principle \(https://gettingbettereveryday.org/2018/10/05/what-you-could-learn-from-barbra-mintos-the-pyramid-principle-2009-172-pages/\)](https://gettingbettereveryday.org/2018/10/05/what-you-could-learn-from-barbra-mintos-the-pyramid-principle-2009-172-pages/)

Your Turn!

 Get ready for your presentations!