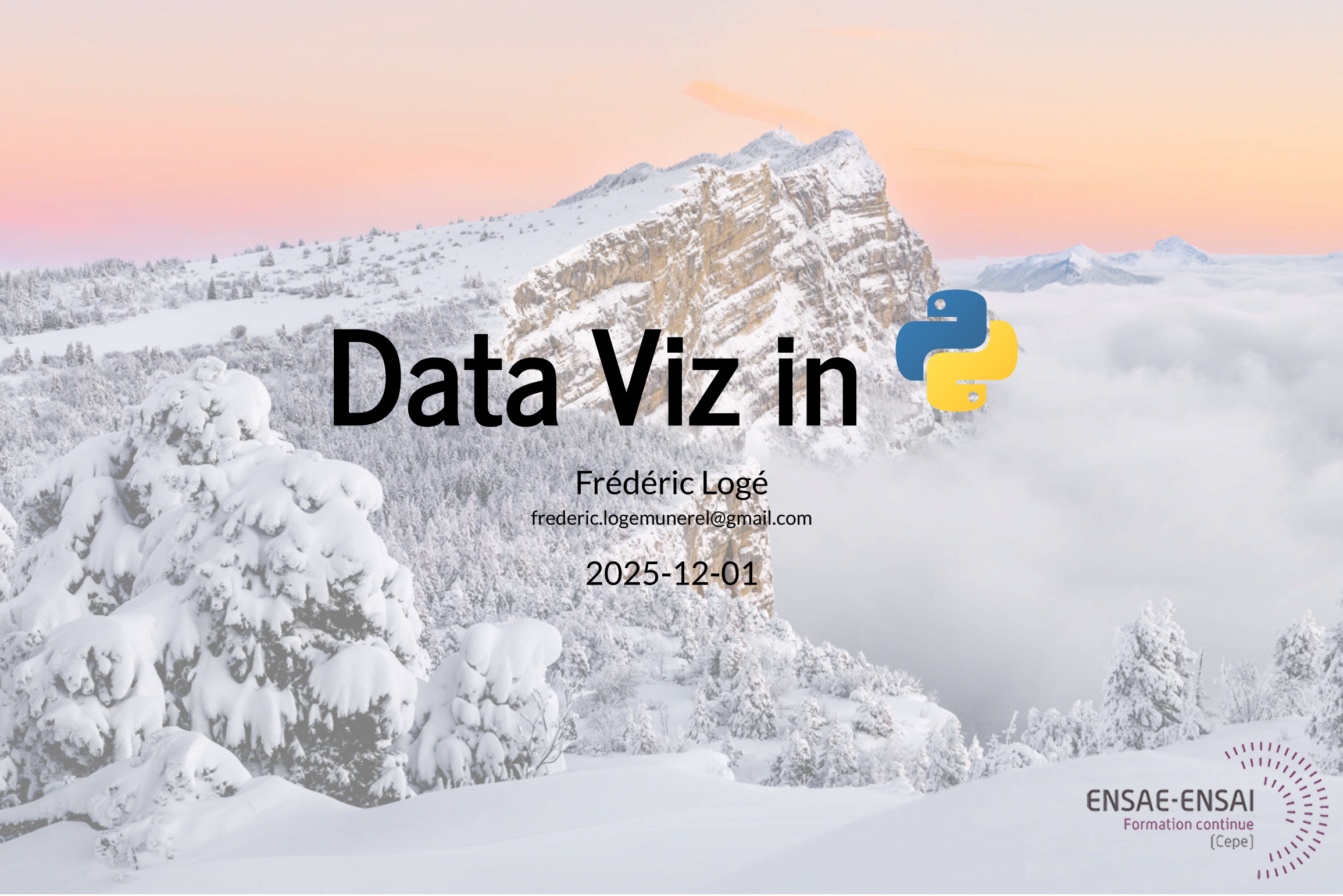


Data Viz in

Frédéric Logé
frederic.logemunerel@gmail.com

2025-12-01

ENSAE-ENSAI
Formation continue
(Cepe)

Program

- Introduction
- Graphics
- Dashboards
- Resources

ENSAE-ENSAI
Formation continue
(Cepe)

Introduction

ENSAE-ENSAI
Formation continue
(Cepe)

Objectives for today's session

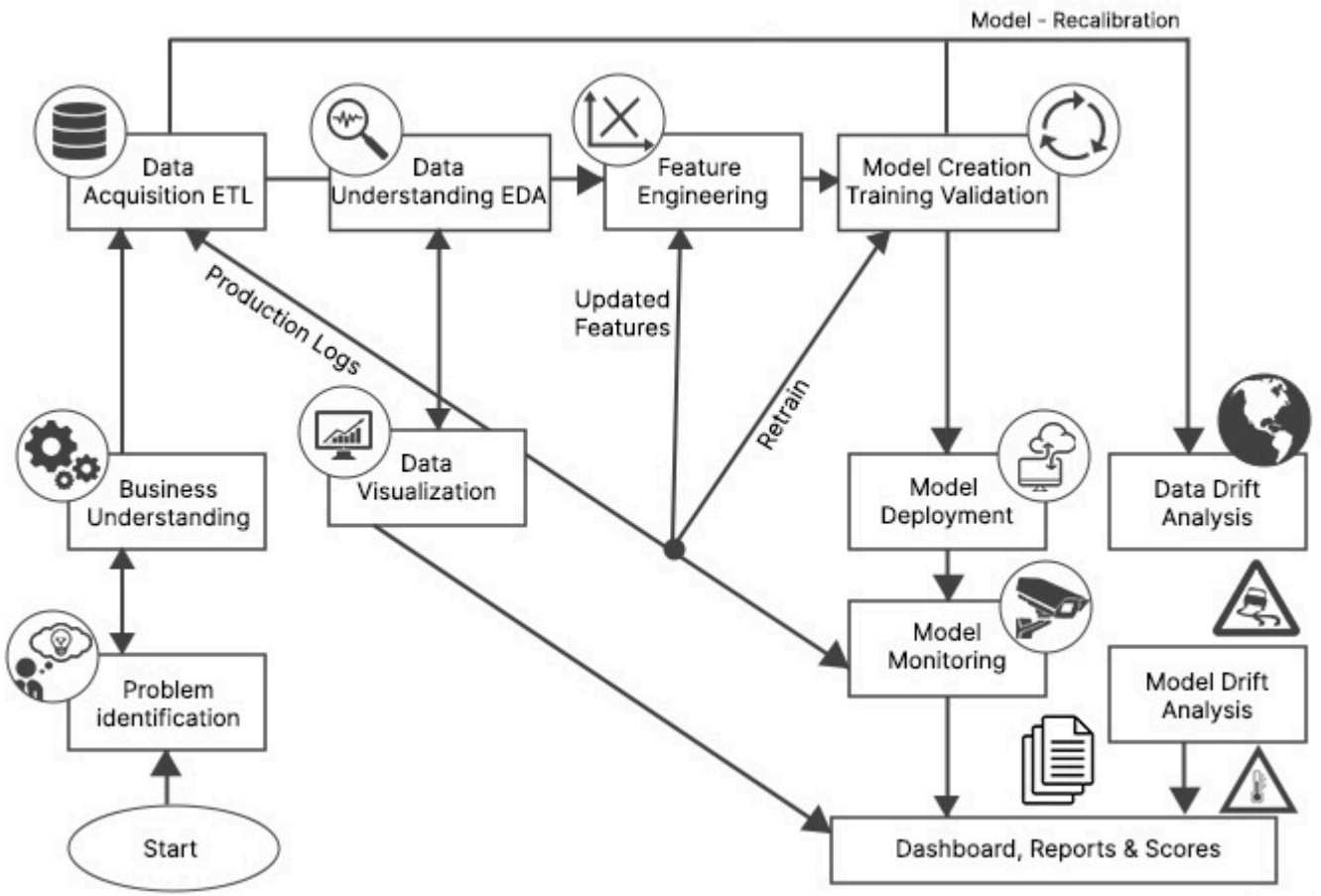
1. End of day, you have a good starter kit for **data viz**, for graphs & dashboards
2. The session should be interactive and ~2/3 of **practice**
3. Opportunity to **discover libraries** you might not use regularly
4. Discuss the importance of **reproducibility, testing & deployment**

🙋 **Any questions ?**

ENSAE-ENSAI
Formation continue
(Cepe)

I have a question 🙋

Where does data visualization fit in this data science life cycle ? what about dashboards ?



Data Science Lifecycle source

ENSAE-ENSAI
Formation continue
(Cepe)

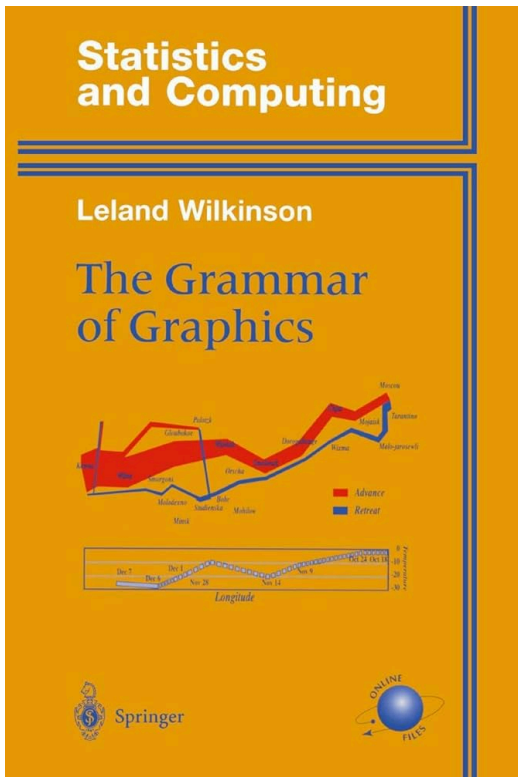
Graphics

ENSAE-ENSAI
Formation continue
(Cepe)

Grammar of Graphics

Grammar of graphics following Hadley Wickham, Chief Scientist @ posit:

1. Data
2. Aesthetics
3. Geometries
4. Facets
5. Statistics
6. Coordinates
7. Theme



Grammar of graphics

ENSAE-ENSAI
Formation continue
(Cepe)

plotnine: a good place to start

```python
import plotnine as pn
url = "https://raw.githubusercontent.com/tidyverse/dplyr/master/data-raw/starwars.csv"
df = pd.read_csv(url)

my_plot = (
  pn.ggplot(data=df)+                                    # Data
  pn.aes(x='height', y='mass', color='eye_color')+       # Aesthetics
  pn.geom_point()+                                       # Geometries
  pn.geom_smooth(method='lm')+
  pn.facet_wrap('gender')+                               # Facets
  pn.scale_x_log()+                                      # Coordinates
  pn.theme(text=element_text(size=10))+                  # Theme
  pn.labs(
    x='Height (cm)', y='Mass (kg)', color='Eye Color',
    title='Star Wars Characters: Height vs Mass'
  )

)

my_plot
```

ENSAE-ENSAI
Formation continue
(Cepe)

Practice Time 

1. Create the plot by adding progressively each code line to understand what happens
2. Expand to other common geometries
3. Improve theming
4. Test other libraries
5. 🍪 / ☕

ENSAE-ENSAI
Formation continue
(Cepe)

Notable packages

Vega-Altair

The problem is Choice

Question: which ones do you typically use, and most importantly, why ?

Deciding factors are often:

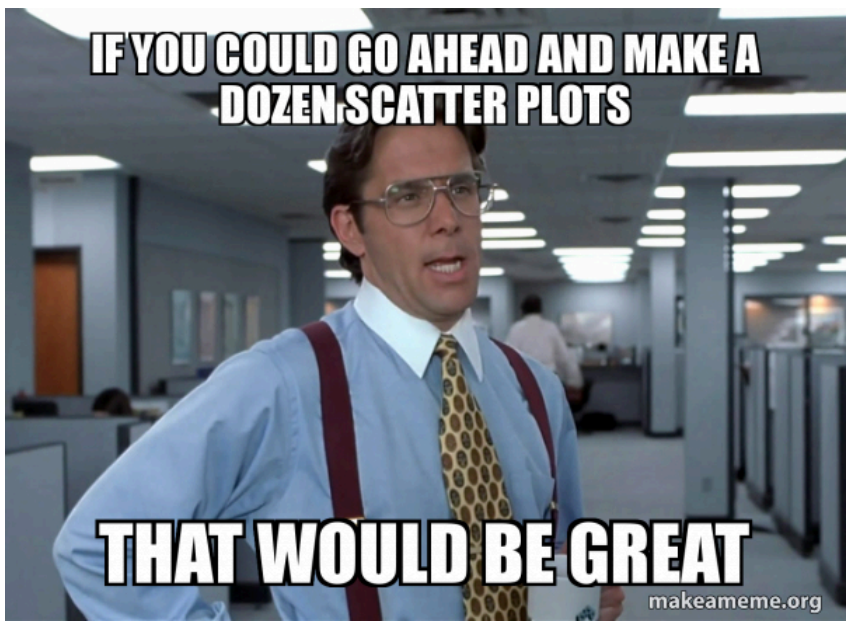
- code readability
- ease of use
- support from the community
- code maturity
- active developer**s**
- aesthetics and personal taste
- personal habit

ENSAE-ENSAI
Formation continue
(Cepe)

Comparing syntax

Choose one library amongst the following that you're not super familiar with already: `seaborn`, `altair`, `plotly`, `bokeh`, … and recreate with this library **two** different visualizations we did using the starwars data.

For the purpose of the exercise, I recommend you rely on the libraries online documentation and, if any trouble occurs, please rely on stackoverflow and co or myself. Let's try to solve this on our own, without AI agent.



ENSAE-ENSAI
Formation continue
(Cepe)

plotnine example

plotnine, our reference is:

```python
import plotnine as pn

my_plot = (
  pn.ggplot(data=df)+
  pn.aes(x='height', y='mass', color='eye_color')+
  pn.geom_point()+
  pn.labs(
    x='Height (cm)', y='Mass (kg)', color='Eye Color',
    title='Star Wars Characters: Height vs Mass'
  )
)

my_plot
```

ENSAE-ENSAI
Formation continue
(Cepe)

`matplotlib` example

is the basic method for viz in python, can mix a lot of methods but it tends to be very verbose even for simple cases (see here for just adding a legend 🙈)

```python
import matplotlib.pyplot as plt

plt.figure()
scatter = plt.scatter(
    df["height"],
    df["mass"],
    c=pd.Categorical(df["eye_color"]).codes,  # convert category → numeric colors
    cmap="tab10",
)

plt.xlabel("Height (cm)")
plt.ylabel("Mass (kg)")
plt.title("Star Wars Characters: Height vs Mass")

# legend
eye_colors = df["eye_color"].astype("category")
handles = [
    plt.Line2D([], [], marker="o", linestyle="", color=scatter.cmap(scatter.norm(code)))
    for code in range(len(eye_colors.cat.categories))
]
plt.legend(handles, eye_colors.cat.categories, title="Eye Color")

plt.show()
```

ENSAE-ENSAI
Formation continue
(Cepe)

`plotly` example

offers immediate interactivity features. Note that there is `plotly graph objects` and `plotly express`.

```
import plotly.express as px

fig = px.scatter(
    df,
    x="height",
    y="mass",
    color="eye_color",
    hover_data={
        "name": True, # adding name
        "homeworld": True, # adding home world
        "height": ":.0f", # formatting height
    },
    title="Star Wars Characters: Height vs Mass"
)

fig.show()
```

ENSAE-ENSAI
Formation continue
(Cepe)

`seaborn` example

is build on top of matplotlib and produces very beautiful outputs

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(6, 4))
sns.scatterplot(
    data=df,
    x="mass",
    y="height",
    hue="eye_color"
)

plt.title("Star Wars Characters: Height vs Mass")
plt.tight_layout()
plt.show()
```

ENSAE-ENSAI
Formation continue
(Cepe)

`altair` example

has an interesting style of variable declaration and produces very beautiful outputs

```
import altair as alt

chart = alt.Chart(df).mark_circle().encode(
    x=alt.X("height", title="Height (cm)"),
    y=alt.Y("mass", title="Mass (kg)"),
    color=alt.Color("eye_color", title="Eye Color"),
    tooltip=["name", "species", "height", "mass", "eye_color"]
).properties(title="Star Wars Characters: Height vs Mass")

chart.interactive()
```

ENSAE-ENSAI
Formation continue
(Cepe)

Specialized packages

For cartography:  and GeoViews

For network representation: `plotly` is actually quite good! `pyvis` is also very nice but no longer maintained following 2022.

Some libraries sometimes hide a lot of treasures, you should spend a lot of time in their docs & galleries!

Question: is there another type of plot you have in mind we might not have covered ?

Dashboards

ENSAE-ENSAI
Formation continue
(Cepe)

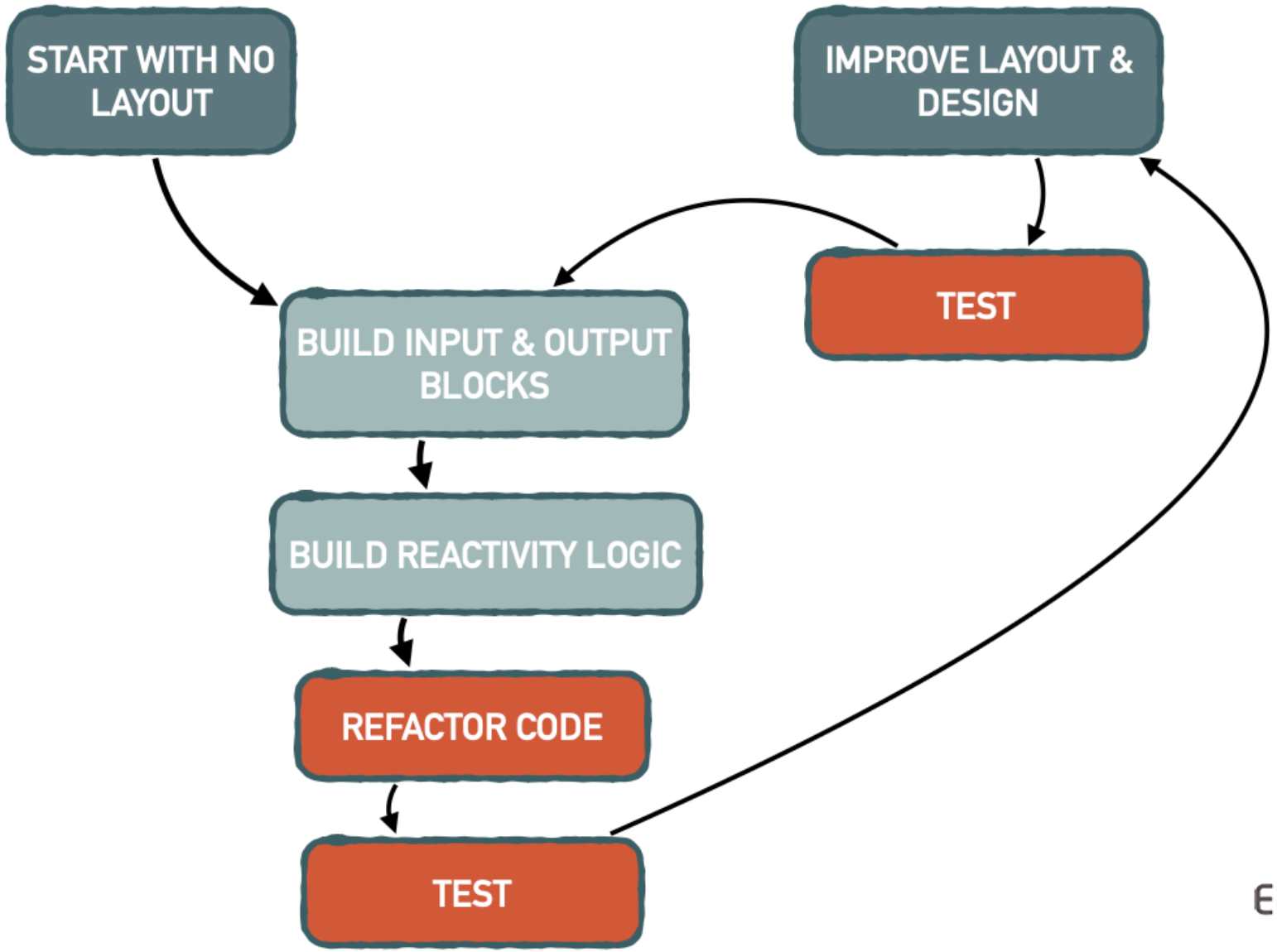
Code structure for data apps

Most libraries help you code in python as wrapper for html, css and JS code. As such, you have essentially the following five code sections to handle:

1. **Inputs** come from the user e.g. data, filter selection, action request
2. **Outputs** are expected to be updated with the provided input e.g. a table is displayed for a given input filter, a graph generated for variables specified in input
3. **Reactivity** is the way of connecting inputs to outputs i.e. making sure that an when an input is changed the proper output is modified
4. **Layout** is how every element is organized on the app e.g. tabs, menus, column pagination
5. **Style** is how every element actually looks when it is displayed (pure CSS behind it)

ENSAE-ENSAI
Formation continue
(Cepe)

General Development process



ENSAE-ENSAI
Formation continue
(Cepe)

shiny: a reliable start base

```python
# in terminal: shiny run --reload basic-app-example-shiny.py
from shiny import App, render, ui
import plotnine as pn
import pandas as pd

df = pd.read_csv("https://raw.githubusercontent.com/tidyverse/dplyr/master/data-raw/starwars.csv")

## user interface (UI) definition
app_ui = ui.page_fixed(                                  # HTML/CSS/JS content
    ui.input_select(                                     # UI Input Element
        id="var", label="Select variable", choices=["height", "mass"]
    ),
    ui.output_plot("hist")                               # UI Output Element
)

## server function provides access to client-side input values
def server(input):                                 # python function
    @render.plot                                   # indicate reactivity + output type
    def hist():
        return (
          pn.ggplot(data=df)+
          pn.aes(x=input.var())+ # accessing input.var() value
          pn.geom_histogram()
        )

app = App(app_ui, server)
```

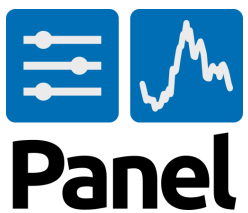
ENSAE-ENSAI
Formation continue
(Cepe)

Practice Time 

1. Run the app locally
2. Add two inputs
3. Add three outputs
4. Improve layout / presentation
5. 🍪 / ☕

ENSAE-ENSAI
Formation continue
(Cepe)

Quick tour of other approaches



Dash example

```python
# in terminal: python basic-app-example-dash.py
import dash
from dash import dcc, html, Input, Output
import plotly.express as px
import pandas as pd

df = pd.read_csv("https://raw.githubusercontent.com/tidyverse/dplyr/master/data-raw/starwars.csv")

app = dash.Dash(__name__)

app.layout = html.Div([
    dcc.Dropdown(id="var", options=["mass", "height"],       # UI Input Element
      value="mass", clearable=False),
    dcc.Graph(id="hist")                                     # UI Output Element
])

@app.callback(                          # callback: update the var selected changes
    Output("hist", "figure"),
    Input("var", "value")
)
def update_hist(selected_var):
    return px.histogram(df, x=selected_var, title=f"Histogram of {selected_var}")

app.run(debug=True, use_reloader=False) # debugger is super useful!
```

ENSAE-ENSAI
Formation continue
(Cepe)

Panel example

```python
# panel serve basic-app-example-panel.py --dev
import panel as pan
import pandas as pd
import plotly.express as px
pan.extension('plotly')
df = pd.read_csv("https://raw.githubusercontent.com/tidyverse/dplyr/master/data-raw/starwars.csv")

var = pan.widgets.Select(                                    # UI Input element
    name="Select variable", options=["mass", "height"], value="mass"
)

def make_plot(selected_var):                    # python function to update output
    return px.histogram(
      df, x=selected_var, title=f"Histogram of {selected_var}"
    )

interactive_plot = pan.bind(                # binding output function to an input
  make_plot, selected_var=var
)

app = pan.Column(                          # Layout
    var,                                   # Input
    pan.pane.Plotly(interactive_plot)      # Output
)

app.servable()
```

ENSAE-ENSAI
Formation continue
(Cepe)

Streamlit example

```python
# in terminal:  streamlit run basic-app-example-streamlit.py

import streamlit as st
import pandas as pd
import plotly.express as px

df = pd.read_csv("https://raw.githubusercontent.com/tidyverse/dplyr/master/data-raw/starwars.csv")

var = st.selectbox(                                    # UI Input
    "Select variable:",
    ["mass", "height"],
    index=0
)

fig = px.histogram(                                    # Output computation
    df,
    x=var,
    title=f"Histogram of {var}"
)

st.plotly_chart(fig, use_container_width=True)      # UI Output
```

ENSAE-ENSAI
Formation continue
(Cepe)

Going further

Code organization: as I'm sure you've noticed, it can get messy very quickly when programming dashboards. Common sense prevails: add comments, clear names, refactor in functions or classes and move code outside.

Deployment: using your app locally for individual usage or POCs is fine, but the ultimate goal is to make it available to internal and external users. The most common approach is to package your code (common approach is to create a Docker image) and make it run on servers either locally or on a cloud-based service. Here's a non-exhaustive list of hosting services: AWS EC2, Hugging Face, Azure, GCP shinyapps.io, Plotly Cloud, Heroku, see more. If you have a Data Engineer friend, now's the time to call them.

What it's not: those libraries were intended to facilitate any Data Scientist's life to be able to show and deploy analyses, statistics and models, by having to not care about HTML, CSS and Javascript. Although you could make a chess game using those libraries, that would be a strange technical choice. If you want to create a static website with no ties to data, again very strange. If you want to make a full-on website, then it's just not the right tool.

ENSAE-ENSAI
Formation continue
(Cepe)

Resources

obliviously not exhaustive 😉

`shiny` API

`panel` docs

`plotly` docs

`plotnine` cheatsheet

`altair` docs

ENSAE-ENSAI
Formation continue
(Cepe)