TP : Découverte de Dash (Front-End) – Fil rouge Dashboard

Elie Beyeler, Kylian Deschamps, Kadir Eramil

Objectif

L'objectif de ce TP est de découvrir la bibliothèque **Dash**, utilisée pour créer des applications web interactives en Python. Nous allons principalement explorer son aspect **Front-End** : la mise en page, les composants graphiques et l'interactivité, à travers un **fil rouge** :

*Construire pas à pas un petit **dashboard de ventes** stylé.*

À la fin de ce TP, vous saurez :
— créer une application Dash avec un thème Bootstrap ;
— utiliser les composants de base (`Input`, `Dropdown`, `Slider`, `Graph`) ;
— intégrer un graphique Plotly à partir d'un `DataFrame pandas` ;
— organiser une page en lignes/colonnes avec `dash-bootstrap-components` ;
— écrire plusieurs *callbacks* pour rendre la page interactive.

1 Mise en place de l'environnement

Avant de commencer, nous allons créer un environnement dédié au TP et installer les bibliothèques nécessaires.

Création d'un environnement virtuel (recommandé)

Dans un terminal, placez-vous dans le dossier où vous souhaitez travailler, puis exécutez :

```
python -m venv venv
```

Activez ensuite l'environnement virtuel :
— Sous Linux / macOS :

```
source venv/bin/activate
```

— Sous Windows :

```
venv\Scripts\activate
```

Installation des dépendances

Installez ensuite les bibliothèques nécessaires :

```
pip install dash plotly dash-bootstrap-components numpy pandas
```

Structure minimale du projet

Créez un fichier `app.py` dans un nouveau dossier, par exemple `tp_dash/`. Tout le code des exercices sera écrit (ou modifié) dans ce fichier.

2 Introduction à Dash

Dash repose sur 3 piliers :
— **Flask** pour la partie serveur (back-end), que l'on n'explorera pas dans ce TP,
— **React.js** pour l'affichage (front-end),
— **Plotly.js** pour les graphiques interactifs.
L'idée principale de Dash est de permettre de créer des applications web en Python, sans avoir à écrire de JavaScript.

Dans ce TP, nous allons construire progressivement un **dashboard de ventes**.

3 Fil rouge : Dashboard de ventes interactif

Nous allons travailler tout au long du TP dans le **même fichier** `app.py`. À chaque exercice, on améliore la même page pour qu'elle ressemble de plus en plus à un vrai dashboard.

Jeu de données utilisé

Nous allons utiliser un petit jeu de données synthétique représentant des ventes mensuelles par région.

```
import pandas as pd

data = {
    "mois":   ["Jan", "Fev", "Mar", "Avr", "Mai", "Juin"] * 3,
    "region": ["Nord"] * 6 + ["Sud"] * 6 + ["Est"] * 6,
    "ventes": [120, 150, 130, 170, 160, 180,
               100, 110, 115, 130, 140, 150,
               90, 105, 95, 120, 125, 135],
    "annee":  [2024] * 18
}

df = pd.DataFrame(data)
```

4 Exercice 1 : Base de l'application + thème Bootstrap

Objectif : Créer une application Dash avec un thème sombre Bootstrap et une structure de base.

Consigne : Dans `app.py`, créez une application Dash avec `dash-bootstrap-components` et un layout minimal contenant :

— un titre centré,
— un court sous-titre décrivant le dashboard,
— un conteneur principal pour les éléments à venir.

```
import dash
from dash import html
import dash_bootstrap_components as dbc

# Cr ation de l'application avec un th me Bootstrap
app = dash.Dash(__name__, external_stylesheets=[dbc.themes.CYBORG])

app.layout = dbc.Container([
    dbc.Row([
        dbc.Col(html.H1("Dashboard de ventes",
                        className="text-center text-primary mt-4 mb
                           -2"))
    ]),
    dbc.Row([
        dbc.Col(html.P(
            "Fil rouge : nous allons enrichir ce dashboard  tape
               par  tape .",
            className="text-center mb-4"
        ))
    ]),
    # Zone principale (que l'on remplira dans les exercices
       suivants)
    dbc.Row([
        dbc.Col(html.Div("Contenu    venir...", id="main-content"))
    ])
], fluid=True)

if __name__ == "__main__":
    app.run(debug=True)
```

Vérifiez que l'application se lance correctement et que le thème sombre est bien appliqué.

5 Exercice 2 : Ajout des données et des indicateurs (KPIs) statiques

Objectif : Afficher des indicateurs clés (KPI) à partir du `DataFrame`.

Consigne :

— Créez le `DataFrame df` (voir plus haut).

— Remplacez le contenu de `id="main-content"` par une ligne de 3 cartes (*cards*) affichant :

1. le total des ventes,
2. le nombre de régions,
3. le meilleur mois (nom du mois avec le plus de ventes globales).

```
from dash import html
import dash_bootstrap_components as dbc
import pandas as pd

# ... cr ation du DataFrame df comme indiqu  plus haut ...

total_ventes = df["ventes"].sum()
nb_regions = df["region"].nunique()
meilleur_mois = df.groupby("mois")["ventes"].sum().idxmax()

app.layout = dbc.Container([
    dbc.Row([
        dbc.Col(html.H1("Dashboard de ventes",
                        className="text-center text-primary mt-4 mb
                            -2"))
    ]),
    dbc.Row([
        dbc.Col(html.P(
            "Fil rouge : tableau de bord des ventes 2024.",
            className="text-center text-light mb-4"
        ))
    ]),
    dbc.Row([
        dbc.Col(dbc.Card([
            dbc.CardHeader("Total des ventes"),
            dbc.CardBody(html.H3(f"{total_ventes}", className="text
                -success"))
        ], className="mb-3"), md=4),
        dbc.Col(dbc.Card([
            dbc.CardHeader("Nombre de r gions"),
            dbc.CardBody(html.H3(f"{nb_regions}", className="text-
                info"))
        ], className="mb-3"), md=4),
        dbc.Col(dbc.Card([
            dbc.CardHeader("Meilleur mois (global)"),
            dbc.CardBody(html.H3(meilleur_mois, className="text-
                warning"))
        ], className="mb-3"), md=4),
    ], className="mb-4", id="kpi-row"),
    # Nous ajouterons d'autres    lments    sous les KPIs
    dbc.Row(id="graph-row")
], fluid=True)
```

À ce stade, les indicateurs sont encore **statiques** (pas de callback).

6 Exercice 3 : Premier graphique (Plotly Express)

Objectif : Ajouter un graphique montrant l'évolution des ventes par mois (toutes régions confondues).

Consigne :

— Importez `plotly.express` sous le nom `px`.

— Créez une figure affichant les ventes totales par mois (barres ou courbe).

— Affichez cette figure dans un composant `dcc.Graph` sous les cartes KPI.

```
import dash
from dash import dcc, html
import dash_bootstrap_components as dbc
import pandas as pd
import plotly.express as px

# ---------- DATA ----------
data = {
    "mois":   ["Jan", "F  v", "Mar", "Avr", "Mai", "Juin"] * 3,
    "region": ["Nord"] * 6 + ["Sud"] * 6 + ["Est"] * 6,
    "ventes": [120, 150, 130, 170, 160, 180,
               100, 110, 115, 130, 140, 150,
               90, 105, 95, 120, 125, 135],
    "annee":  [2024] * 18
}

df = pd.DataFrame(data)

# ---------- FIGURE PAR D FAUT ----------
df_mois = df.groupby("mois", as_index=False)["ventes"].sum()

fig_ventes_globales = px.bar(
    df_mois,
    x="mois",
    y="ventes",
    title="Ventes totales par mois (toutes r gions)"
)

# ---------- APP ----------
app = dash.Dash(__name__, external_stylesheets=[dbc.themes.CYBORG])

total_ventes = df["ventes"].sum()
nb_regions = df["region"].nunique()
meilleur_mois = df.groupby("mois")["ventes"].sum().idxmax()

app.layout = dbc.Container([
    dbc.Row([
        dbc.Col(html.H1("Dashboard de ventes",
                        className="text-center text-primary mt-4 mb
                           -2"))
    ]),
    dbc.Row([
        dbc.Col(html.P(
```

```
            "Fil rouge : tableau de bord des ventes 2024.",
            className="text-center mb-4"
        ))
    ]),

    # KPIs
    dbc.Row([
        dbc.Col(dbc.Card([
            dbc.CardHeader("Total des ventes"),
            dbc.CardBody(html.H3(f"{total_ventes}", className="text
                -success"))
        ], className="mb-3"), md=4),
        dbc.Col(dbc.Card([
            dbc.CardHeader("Nombre de regions"),
            dbc.CardBody(html.H3(f"{nb_regions}", className="text-
                info"))
        ], className="mb-3"), md=4),
        dbc.Col(dbc.Card([
            dbc.CardHeader("Meilleur mois (global)"),
            dbc.CardBody(html.H3(meilleur_mois, className="text-
                warning"))
        ], className="mb-3"), md=4),
    ], className="mb-4", id="kpi-row"),

    # GRAPH
    dbc.Row([
        dbc.Col(dcc.Graph(id="graph-ventes", figure=
            fig_ventes_globales), md=12)
    ], id="graph-row")
], fluid=True)

if __name__ == "__main__":
    app.run(debug=True)
```

Vous devez obtenir un graphique simple mais déjà lisible.

7 Exercice 4 : Filtrer par région (1er callback)

Objectif : Ajouter un menu déroulant pour filtrer le graphique par région. C'est notre **premier callback**.

Consigne :

— Ajoutez un `dcc.Dropdown` au-dessus du graphique, avec les options : `"Toutes"` + une option par région.

— Créez un callback qui met à jour la figure `id="graph-ventes"` en fonction de la région choisie.

```
import dash
from dash import dcc, html
import dash_bootstrap_components as dbc
from dash.dependencies import Input, Output
import pandas as pd
```

```
import plotly.express as px

# ---------- DATA ----------
data = {
    "mois":    ["Jan", "F  v", "Mar", "Avr", "Mai", "Juin"] * 3,
    "region": ["Nord"] * 6 + ["Sud"] * 6 + ["Est"] * 6,
    "ventes": [120, 150, 130, 170, 160, 180,
               100, 110, 115, 130, 140, 150,
               90, 105, 95, 120, 125, 135],
    "annee":   [2024] * 18
}

df = pd.DataFrame(data)

# ---------- APP ----------
app = dash.Dash(__name__, external_stylesheets=[dbc.themes.CYBORG])

total_ventes = df["ventes"].sum()
nb_regions = df["region"].nunique()
meilleur_mois = df.groupby("mois")["ventes"].sum().idxmax()

# options du dropdown de r gion
regions = sorted(df["region"].unique())
options_region = [{"label": "Toutes les r gions", "value": "Toutes
   "}] + \
    [{"label": r, "value": r} for r in regions]

app.layout = dbc.Container([
    dbc.Row([
        dbc.Col(html.H1("Dashboard de ventes",
                        className="text-center text-primary mt-4 mb
                           -2"))
    ]),
    dbc.Row([
        dbc.Col(html.P(
            "Fil rouge : tableau de bord des ventes 2024.",
            className="text-center mb-4"
        ))
    ]),

    # KPIs
    dbc.Row([
        dbc.Col(dbc.Card([
            dbc.CardHeader("Total des ventes"),
            dbc.CardBody(html.H3(f"{total_ventes}", className="text
               -success"))
        ], className="mb-3"), md=4),
        dbc.Col(dbc.Card([
            dbc.CardHeader("Nombre de r gions"),
            dbc.CardBody(html.H3(f"{nb_regions}", className="text-
               info"))
```

```
        ], className="mb-3"), md=4),
        dbc.Col(dbc.Card([
            dbc.CardHeader("Meilleur mois (global)"),
            dbc.CardBody(html.H3(meilleur_mois, className="text-
                warning"))
        ], className="mb-3"), md=4),
    ], className="mb-4", id="kpi-row"),

    # DROPDOWN DE REGION
    dbc.Row([
        dbc.Col(
            dcc.Dropdown(
                id="dropdown-region",
                options=options_region,
                value="Toutes",       # valeur initiale
                clearable=False
            ),
            md=4
        )
    ], className="mb-3"),

    # GRAPH
    dbc.Row([
        dbc.Col(dcc.Graph(id="graph-ventes"), md=12)
    ], id="graph-row")
], fluid=True)


# ---------- CALLBACK : met    jour le graph selon la r gion
    ----------
@app.callback(
    Output("graph-ventes", "figure"),
    Input("dropdown-region", "value")
)
def update_graph(region):
    if region == "Toutes":
        dff = df
        titre = "Ventes totales par mois (toutes r gions)"
    else:
        dff = df[df["region"] == region]
        titre = f"Ventes par mois ({region})"

    df_mois_region = dff.groupby("mois", as_index=False)["ventes"].
        sum()

    fig = px.bar(
        df_mois_region,
        x="mois",
        y="ventes",
        title=titre
    )
```

```
    return fig


if __name__ == "__main__":
    app.run(debug=True)
```

Callback n°1 : `update_graph`.

8 Exercice 5 : KPIs dynamiques (2e et 3e callbacks)

Objectif : Faire en sorte que les cartes KPI s'adaptent au filtre de région.
Consigne :

— Ajoutez des `id` sur les éléments contenant les valeurs des KPI (par exemple `"kpi-total"`, `"kpi-meilleur-mois"`).

— Créez deux callbacks supplémentaires qui mettent à jour :

1. le total des ventes (en fonction de la région choisie) ;
2. le meilleur mois pour cette région (ou global si `"Toutes"`).

```
import dash
from dash import dcc, html
import dash_bootstrap_components as dbc
from dash.dependencies import Input, Output
import pandas as pd
import plotly.express as px

# ---------- DATA ----------
data = {
    "mois":   ["Jan", "F  v", "Mar", "Avr", "Mai", "Juin"] * 3,
    "region": ["Nord"] * 6 + ["Sud"] * 6 + ["Est"] * 6,
    "ventes": [120, 150, 130, 170, 160, 180,
               100, 110, 115, 130, 140, 150,
               90, 105, 95, 120, 125, 135],
    "annee":  [2024] * 18
}

df = pd.DataFrame(data)

# ---------- APP ----------
app = dash.Dash(__name__, external_stylesheets=[dbc.themes.CYBORG])

nb_regions = df["region"].nunique()

# ---------- DROPDOWN OPTIONS ----------
regions = sorted(df["region"].unique())
options_region = [{"label": "Toutes les r gions", "value": "Toutes
   "}] + \
    [{"label": r, "value": r} for r in regions]

# ---------- LAYOUT ----------
app.layout = dbc.Container([
```

```
    dbc.Row([
        dbc.Col(html.H1("Dashboard de ventes",
                        className="text-center text-primary mt-4 mb
                            -2"))
    ]),
    dbc.Row([
        dbc.Col(html.P(
            "Fil rouge : tableau de bord des ventes 2024.",
            className="text-center mb-4"
        ))
    ]),

    # KPIs      maintenant avec des id !!!
    dbc.Row([
        dbc.Col(dbc.Card([
            dbc.CardHeader("Total des ventes"),
            dbc.CardBody(html.H3(id="kpi-total", className="text-
                success"))
        ], className="mb-3"), md=4),

        dbc.Col(dbc.Card([
            dbc.CardHeader("Nombre de r gions"),
            dbc.CardBody(html.H3(f"{nb_regions}", className="text-
                info"))
        ], className="mb-3"), md=4),

        dbc.Col(dbc.Card([
            dbc.CardHeader("Meilleur mois"),
            dbc.CardBody(html.H3(id="kpi-meilleur-mois", className=
                "text-warning"))
        ], className="mb-3"), md=4),
    ], className="mb-4", id="kpi-row"),

    # DROPDOWN REGION
    dbc.Row([
        dbc.Col(
            dcc.Dropdown(
                id="dropdown-region",
                options=options_region,
                value="Toutes",
                clearable=False
            ),
            md=4
        )
    ], className="mb-3"),

    # GRAPH
    dbc.Row([
        dbc.Col(dcc.Graph(id="graph-ventes"), md=12)
    ])
], fluid=True)
```

```python
# ---------- CALLBACK 1 : mettre    jour le graphique ----------
@app.callback(
    Output("graph-ventes", "figure"),
    Input("dropdown-region", "value")
)
def update_graph(region):
    if region == "Toutes":
        dff = df
        titre = "Ventes totales par mois (toutes r gions)"
    else:
        dff = df[df["region"] == region]
        titre = f"Ventes par mois ({region})"

    df_mois = dff.groupby("mois", as_index=False)["ventes"].sum()

    fig = px.bar(df_mois, x="mois", y="ventes", title=titre)
    return fig


# ---------- CALLBACK 2 : mettre    jour KPI TOTAL ----------
@app.callback(
    Output("kpi-total", "children"),
    Input("dropdown-region", "value")
)
def update_kpi_total(region):
    if region == "Toutes":
        dff = df
    else:
        dff = df[df["region"] == region]
    return dff["ventes"].sum()


# ---------- CALLBACK 3 : mettre    jour KPI MEILLEUR MOIS
    ----------
@app.callback(
    Output("kpi-meilleur-mois", "children"),
    Input("dropdown-region", "value")
)
def update_kpi_best_month(region):
    if region == "Toutes":
        dff = df
    else:
        dff = df[df["region"] == region]

    return dff.groupby("mois")["ventes"].sum().idxmax()


# ---------- RUN ----------
if __name__ == "__main__":
```

```
    app.run(debug=True)
```

Callbacks n°2 et n°3 : `update_kpi_total` et `update_kpi_meilleur_mois`.

9 Exercice 6 : Détail d'un point cliqué (4e callback)

Objectif : Afficher des informations détaillées sur une barre cliquée du graphique.
Consigne :

— Ajoutez un `html.Div` sous le graphique, avec par exemple `id="detail-point"`.
— Créez un callback qui prend en entrée `clickData` du graphique et affiche un message du type : `"Vous avez cliqué sur le mois X : Y ventes."`

```
# Dans le layout, sous le graphique :
dbc.Row([
    dbc.Col(dcc.Graph(id="graph-ventes"), md=12)
], className="mb-3"),
dbc.Row([
    dbc.Col(html.Div(
        "Cliquez sur une barre pour voir le d tail.",
        id="detail-point",
        className="text-light"
    ))
]),

# Callback utilisant clickData
@app.callback(
    Output("detail-point", "children"),
    Input("graph-ventes", "clickData")
)
def update_detail(clickData):
    if clickData is None:
        return "Cliquez sur une barre pour voir le d tail."

    point = clickData["points"][0]
    mois = point["x"]
    ventes = point["y"]
    return f"Vous avez cliqu  sur le mois {mois} : {ventes} ventes
       ."
```

Callback n°4 : `update_detail`.

10 Exercice 7 : Finition du dashboard (personnalisation du graphique)

Objectif : Ajouter un petit contrôle de présentation pour rendre le dashboard plus "stylé".
Exemple de consigne :

— Ajoutez des `dcc.RadioItems` permettant de choisir le type de graphique :
  — `"bar"` (barres),

— `"line"` (courbe).

— Modifiez le callback `update_graph` de l'exercice 4 pour prendre également en entrée ce choix de type, et générer soit un `px.bar`, soit un `px.line`.

```
# Dans le layout, dans la m me card que le dropdown ou     c  t   :
dcc.RadioItems(
    id="radio-type-graph",
    options=[
        {"label": "Barres", "value": "bar"},
        {"label": "Courbe", "value": "line"}
    ],
    value="bar",
    inline=True,
    className="mt-2"
)

# Adapter le callback de l'exercice 4 :
@app.callback(
    Output("graph-ventes", "figure"),
    Input("dropdown-region", "value"),
    Input("radio-type-graph", "value")
)
def update_graph(region, graph_type):
    if region == "Toutes":
        dff = df
        titre = "Ventes par mois (toutes r gions)"
    else:
        dff = df[df["region"] == region]
        titre = f"Ventes par mois ({region})"

    df_mois_region = dff.groupby("mois", as_index=False)["ventes"].
       sum()

    if graph_type == "bar":
        fig = px.bar(df_mois_region, x="mois", y="ventes", title=
           titre)
    else:
        fig = px.line(df_mois_region, x="mois", y="ventes", title=
           titre)

    return fig
```

À ce stade, vous avez un dashboard complet :

— thème sombre agréable,

— indicateurs dynamiques,

— graphique filtrable par région,

— détail au clic,

— choix du type de visualisation.

Conclusion

À travers ce fil rouge, vous avez :
— mis en place une application Dash avec Bootstrap,
— manipulé un `DataFrame pandas` pour alimenter votre interface,
— construit un graphique Plotly interactif,
— utilisé au moins **quatre callbacks** pour relier les composants entre eux,
— obtenu un **dashboard de ventes** complet et relativement stylé.

Vous disposez maintenant des bases pour créer vos propres dashboards interactifs en Python avec Dash.