La bibliothèque de la semaine
Création d'applications Web avec `dash`

Laurent Bataille

Vendredi 24/04

Traditionnellement, la création de ce genre d'application requiert la connaissance de Javascript. Cependant il est souvent plus simple de travailler à partir d'un seul langage, il est possible aujourd'hui de coder ce genre d'applications en se passant de ce langage, le langage `R` propose par exemple `Shiny`, Python offre différentes possibilités, notamment les extensions `boked`, `django` et `dash`. Pour des raisons de prise en main, `dash` est l'option proposée dans ce tutoriel. Dash est gratuit et OpenSource, sachez cependant qu'il existe une version monétisée pour les entreprises. Les bibliothèques se chargent via les commandes suivante :

```
import dash
import dash_core_components as dcc
import dash_html_components as html
```

`dash_core_components` permet de gérer directement des graphiques à partir de `dash` et d'inclure par exemple des boutons ou des éléments permettant d'interagir avec l'application. `dash_html_components` est une extension permettant d'incorporer des familles de code html prédéfinie dans l'application.

Pour illustrer ce tutoriel, réalisons une animation simple à propos d'un thème d'actualité, combinant différent types de données :

— Le cours de l'action de Kimberly-Clark [1] et de Store Brand [2], NATCO [3], Netflix et Alpha Pro Tech [4] .
— Le taux de change de l'Euro, de la livre Sterling et du Yuan par rapport au Dollar.
— Les cours de l'or, du Bitcoin, du baril de pétrole brut.
— L'indice Google Trends pour la recherche « Coronavirus ».

1. La plus grande entreprise américaine produit principalement des produits de consommation destinés à l'hygiène personnelle et aux soins à partir de papier transformé à l'état de base.
2. Un des acteurs principaux du secteur de la grande distribution détenant une part non-négligeable des parts de marcher dans la vente de pâtes
3. Une société pharmaceutique indienne qui fabrique des formulations pharmaceutiques finies et des ingrédients pharmaceutiques actifs, notamment la désormais célèbre chloroquine
4. Entreprise produisant de l'équipement médical, notamment des masques en Amérique du Nord

Le but de l'application est de visualiser ces différentes grandeurs. Elle contiendra un champ pour sélectionner une fenêtre temporelle avec une date de départ et une date de fin. Un menu déroulant proposant de sélectionner les variables en ordonnée (deux axes) et en abscisse (par défaut la date).

Pour commencer, chargeons les extensions et les données dans l'espace de travail et filtrons les trous en interpolant les valeurs manquantes jour par jour.

```
import pandas as pd #Gestion des tableaux
import numpy as np  #Maths...
from scipy.interpolate import CubicSpline   #Interpolation des fonctions avant données manquantes
import os                               #Gestion des dossiers
from datetime import datetime as dt #Gestion des dates


folder = "./dataCov/"
listSubFolder = os.listdir(folder)


filesM = pd.concat([pd.DataFrame(["Netflix","Livre Sterling","Storebrand ASA","Pétrole brut","Kimberly-Clark","Bitcoin","Google Trends - Covid","Euro","Yuan","NATCO Pharma","Alpha Pro Tech","Or","Mc Donalds"]),
                    pd.DataFrame(["NFLX.csv","GBPUSD=X.csv","STB.OL.csv","BNO.csv","KMB.csv","BTC-USD.csv","covid-trends.csv","EURUSD=X.csv","CNYUSD=X.csv","NATCOPHARM.NS.csv","APT.csv","GOLD_2020-04-24.csv","MCD.csv"]),
                    pd.DataFrame(["USD","USD","USD","USD","USD","USD","PRC","USD","USD","USD","USD","USD","USD"])], axis=1)
filesM.columns = ["CONTENT","FILE","UNITS"]
lFunc = []
for k in range(len(filesM)):

    datak = pd.read_csv(folder + filesM.iloc[k]["FILE"], delimiter=',', sep='.',header=0)
    if (filesM.iloc[k]["CONTENT"] != "Or"):
        datak["Date"] = pd.to_datetime(datak["Date"])
    else:
        # Datasheet où la date est écrite de faÃ§on différente
        datak["Date"] = pd.to_datetime(datak["Date"],dayfirst=True)

    datak.sort_values(by=["Date"])
    datak = datak.reset_index(drop=True)
    # Date de référence premier janvier 2019
    tk = (((np.array(datak["Date"] - dt(2019,1,1,0,0,0))).astype(float))/(86400*1e9))

    if (filesM.iloc[k]["UNITS"] == "USD"):
        fk = np.array(datak["Open"])
    else:
        fk = np.array(datak["Score"])

    flagk = np.isfinite(fk) & np.isfinite(tk)
```

```
    tk =      tk[flagk]
    fk = fk[flagk]

    lFunc = lFunc + [CubicSpline(tk,fk)]
```

Dash offre la possibilité d'utiliser des styles prédéfinis pour les différents éléments de l'application, il est possible d'avoir un aperçu dans le descriptif des styles de Dash. Pour créer une application en chargeant ces styles prédéfinis, contenus dans un fichier css, utilisez la commande suivante :

```
    external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
    # __name__ a pour valeur par défaut main... Il faudra s'en préoccuper
     si des applis définies dans plusieurs fichiers sont en présence...
    app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
    if __name__ == '__main__':
        app.run_server(debug=True)
```

1 Créer une zone de texte

Commençons par un exemple minimaliste, en ajoutant du texte.

```
app.layout = html.Div([html.H3('Titre de ma zone de texte'),html.H6('
    Contenu de ma zone de texte')])

if __name__ == '__main__':
    app.run_server(debug=True)
```

`html.Div` est une commande créant la zone, elle prend en argument une liste, `html.H3` et `html.H6` définisse la taille des différents éléments de la liste [5]. La dernière commande précise le mode de fonctionnement. Pour exécuter l'application, utilisez la console de votre système d'exploitation, en vous positionant préalablement dans le dossier adéquat et lancez le code. Un message s'affiche sur votre console, renseignant la page http://127.0.0.1:8050/. En ouvrant cette page dans votre navigateur préféré, la magie s'opère : votre toute première application prend vie !

2 Affichage du cours de Kimberly-Clark en fonction de la date

```
start_date = dt(2019, 11,1)
end_date = dt(2020, 4, 23)
dl = np.arange(str(start_date),str(end_date), dtype='datetime64[D]')
tl = (dl - np.datetime64("2019-01-01")).astype(float)
vall = lFunc[((filesM["CONTENT"]=='Kimberly-Clark').tolist()).index(True)
    ](tl)
pdl = pd.DataFrame({'Date':dl,'Kimberly-Clark':vall})


app.layout = html.Div([html.H3('Notre premier graphique'),
    dcc.Graph(
```

5. `html.H1` défini la plus grande police, tandis que `html.H6` est associée à la plus petite

```
        id='pltComp',
        figure={
            'data': [
                {'x': pdl['Date'], 'y': pdl['Kimberly-Clark'],'mode':'
    markers',
                'opacity':0.7,'marker':{'size': 15,'line': {'width': 0.5,
     'color': 'white'}}}],
            'layout': dict(
                xaxis={'title': 'Date'},
                yaxis={'title': 'Kimberly-Clark'},
                margin={'l': 40, 'b': 40, 't': 10, 'r': 10},
                hovermode='closest'
            )
        }
    )]

if __name__ == '__main__':
    app.run_server(debug=True)
```

Le champ `id` permet de nommer le composant. La gestion les graphiques est réalisé au moyen de la bibliothèque `dash_core_components`.

3 Affichage du cours de Kimberly-Clark et de l'indice Google Trend pour la recherche Coronavirus en fonction de la date

Pour mettre en perspective le cours de l'action de Kimberly-Clark et la crise du coronavirus, observons cette valeur et affichons simultanément l'indice associé au volume journalier de recherche sur Google du terme "Coronavirus" sur un second axe.

```
start_date = dt(2019, 11,1)
end_date = dt(2020, 4, 23)
dl = np.arange(str(start_date),str(end_date), dtype='datetime64[D]')
tl = (dl - np.datetime64("2019-01-01")).astype(float)
vall = lFunc[((filesM["CONTENT"]=='Kimberly-Clark').tolist()).index(True)
    ](tl)
GTI = lFunc[((filesM["CONTENT"]=="Google Trends - Covid").tolist()).index
    (True)](tl)
GTI[GTI<0] = 0
GTI[GTI>100] = 100
pdl = pd.DataFrame({'Date':dl,'Kimberly-Clark':vall,'Google Trends -
    Covid':GTI})

app.layout = html.Div([html.H3('Notre premier graphique'),
    dcc.Graph(
        id='pltComp',
        figure={
            'data': [
                {'x': pdl['Date'], 'y': pdl['Kimberly-Clark'],'mode':'
    markers',
                'opacity':0.7,'marker':{'size': 15,'line': {'width': 0.5,
     'color': 'white'}}},
                {'x': pdl['Date'], 'y': pdl['Google Trends - Covid'],'
    mode':'markers',yaxis='y2',
```

```
                'opacity':0.7,'marker':{'size': 15,'line': {'width': 0.5,
    'color': 'white'}}}

                ],
            'layout': dict(

                xaxis={'title': 'Date'},
                yaxis={
                'title': 'Kimberly-Clark',
                'type': 'linear',
                'anchor': 'x',
                'overlaying' : 'y2',
                'range' : [0,np.amax(vall)]},
                yaxis2={
                    'title': "%",
                    'type': 'linear',
                    'side': 'right',
                    'anchor': 'x',
                    'range' : [0,100],
                    'showgrid' : False},
                margin={'l': 40, 'b': 40, 't': 10, 'r': 10},
                hovermode='closest'
            )
        }
    )]

if __name__ == '__main__':
    app.run_server(debug=True)
```

Cet exemple est proche du précédent mais a le mérite de généraliser la manipulation des figures.

4 Boutons de sélection de la fenêtre temporelle

Jusqu'ici les différentes versions de l'application ne permettent pas à l'utilisateur d'interagir avec celle-ci, uniquement de jeter un regard émerveiller sur les prodigieuses données affichées à l'écran. C'est ici que la notion de fonction de rappel intervient. Il est possible de créer des fonctions capables de mettre à jour les différentes parties de l'application. Rajoutez dès maintenant deux bibliothèques au début de votre script, elles seront utilisées pour coder les composantes de la fonction de rappel.

```
from dash.dependencies import Input, Output
```

Premièrement, il est nécessaire de rajouter un sélecteur de période, via la commande `dcc.DatePickerRange` et de spécifier un id à cette composante.

```
app.layout = html.Div([html.H3('Mon second graphique'),
                       html.Div([
                        html.H6('Choix de la période'),
                        dcc.DatePickerRange(
                        id='my_date_picker',
                        min_date_allowed=dt(2019, 4, 25),
```

```
                        max_date_allowed=dt(2020, 4, 23),
                        start_date=dt(2019, 11,1),
                        end_date=dt(2020, 4, 23))
                        ]),
                        html.Div([
                        dcc.Graph(
                            id='pltComp')])])
```

On remarque que la composante graphique est actuellement vide. Le graphique sera affiché via la fonction de rappel. Il est ensuite nécessaire de quelle composante de l'interface est mise à jour et quelles sont les arguments attendus en entrée et en sortie, ainsi que l'ordre utilisé. La fonction de mise à jour est définie à la suite de la fonction de rappel.

```

@app.callback(
    Output('pltComp', 'figure'),
    [Input('my_date_picker', 'start_date'),
     Input('my_date_picker', 'end_date')])
def update_graph(start_date,end_date):

    dl = np.arange(str(start_date),str(end_date), dtype='datetime64[D]')
    tl = (dl - np.datetime64("2019-01-01")).astype(float)
    vall = lFunc[((filesM["CONTENT"]=='Kimberly-Clark').tolist()).index(
    True)](tl)

    GTI = lFunc[((filesM["CONTENT"]=="Google Trends - Covid").tolist()).
    index(True)](tl)
    GTI[GTI<0] = 0
    GTI[GTI>100] = 100


    pdl = pd.DataFrame({'Date':dl,'Kimberly-Clark':vall,'Google Trends -
    Covid':GTI})
    return {
        'data': [
                {'x': pdl['Date'], 'y': pdl['Kimberly-Clark'],'mode':'
    markers',
                'opacity':0.7,'marker':{'size': 15,'line': {'width': 0.5,
     'color': 'white'}},'name':'Kimberly-Clark'},
                {'x': pdl['Date'], 'y': pdl['Google Trends - Covid'],'
    mode':'markers','yaxis':'y2',
                'opacity':0.7,'marker':{'size': 15,'line': {'width': 0.5,
     'color': 'white'}},'name':'Google Trends - Covid'}
                 ],
        'layout': dict(
            xaxis={
                'title': 'Date',
                'range':[start_date,end_date]
            },
            yaxis={
                'title': 'Kimberly-Clark [$]',
                'type': 'linear',
                'anchor': 'x',
                'overlaying' : 'y2',
```

```
                'range' : [0,np.amax(vall)]
            },
            yaxis2={
                'title': "%",
                'type': 'linear',
                'side': 'right',
                'range' : [0,100],
                'anchor': 'x',
                'showgrid' : False
            },
            margin={'l': 40, 'b': 40, 't': 10, 'r': 0},
            hovermode='closest'
        )
    }
```

5 Sélection des variables des différents axes

Enfin il est possible d'ajouter une liste à cocher permettant la sélection des variables de l'axe y, ce qui peut être réalisé via la fonction `dcc.Checklist`.

```
app.layout = html.Div([
    html.Div([
        html.Div([
            html.H6('Choix de la période'),
            dcc.DatePickerRange(
            id='my_date_picker',
            min_date_allowed=dt(2019, 4, 25),
            max_date_allowed=dt(2020, 4, 23),
            start_date=dt(2019, 11,1),
            end_date=dt(2020, 4, 23))
            ], className="two columns"),
        html.Div([
            html.H6('Ordonnée'),
            dcc.Checklist(
                id='vary',
                options=[
                    {'label':'Kimberly-Clark', 'value': 'Kimberly-Clark'
},
                    {'label':'Storebrand ASA','value':'Storebrand ASA'},
                    {'label':'NATCO Pharma','value':'NATCO Pharma'},
                    {'label':'Alpha Pro Tech','value':'Alpha Pro Tech'},
                    {'label':'Netflix', 'value': 'Netflix'},
                    {'label':'Mc Donalds', 'value': 'Mc Donalds'},
                    {'label':'Pétrole brut','value':'Pétrole brut'},
                    {'label':'Or','value':'Or'},
                    {'label':'Bitcoin','value':'Bitcoin'},
                    {'label':'Euro','value':'Euro'},
                    {'label':'Yuan','value':'Yuan'},
                    {'label':'Livre Sterling','value':'Livre Sterling'}
                ],
                value=['Kimberly-Clark', 'NATCO Pharma'],
                labelStyle = {'display': 'inline-block'}
```

```
            )
        ], className="four columns"),
    ], className="row"),
    dcc.Graph(
        id='pltComp'
    )
])
```

C'est également une opportunité d'introduire comment gérer plusieurs courbes sur un même axe. Le formalisme est similaire au cas d'une seule courbe, mais il faut structurer l'information sous forme d'une liste.

```
@app.callback(
    Output('pltComp', 'figure'),
    [Input('my_date_picker', 'start_date'),
     Input('my_date_picker', 'end_date'),
     Input('vary', 'value')])
def update_graph(start_date,end_date,vary):
    dl = np.arange(str(start_date),str(end_date), dtype='datetime64[D]')
    #dList = dl.astype(dt)

    tl = (dl - np.datetime64("2019-01-01")).astype(float)

    GTI = lFunc[((filesM["CONTENT"]=="Google Trends - Covid").tolist()).
    index(True)](tl)
    GTI[GTI<0] = 0
    GTI[GTI>100] = 100
    pdl = pd.DataFrame({'Date':dl,'Google Trends - Covid':GTI})


    for varys in vary:
        if (varys!="Google Trends - Covid"):
            vall = lFunc[((filesM["CONTENT"]==varys).tolist()).index(True
    )](tl)
            vall[vall<0] = 0
            pdl =  pd.concat([pdl,pd.DataFrame({varys:vall})],axis=1)

    rmax = pdl[vary].max(numeric_only=True)

    curves = []

    for varys in vary:
        curves.append(dict(
        x=pdl['Date'],
        y=pdl[varys],
        mode='markers',
        opacity=0.7,
        secondary_y=False,
        marker={
            'size': 15,
            'line': {'width': 0.5, 'color': 'white'}
        },
        name=varys))

```

```
41
42      curves.append(dict(
43      x=pdl['Date'],
44      y=pdl['Google Trends - Covid'],
45      yaxis='y2',
46      mode='markers',
47      opacity=0.7,
48      secondary_y=True,
49      marker={
50          'size': 15,
51          'line': {'width': 0.5, 'color': 'white'}
52      },
53      name='Google Trends - Covid'))
54
55
56
57      return {
58          'data': curves,
59          'layout': dict(
60              xaxis={
61                  'title': 'Date',
62                  'range': [start_date,end_date]
63              },
64              yaxis={
65                  'title': '$',
66                  'type': 'linear',
67                  'anchor': 'x',
68                  'overlaying' : 'y2',
69                  'range' : [0,rmax]
70              },
71              yaxis2={
72                  'title': "%",
73                  'type': 'linear',
74                  'side': 'right',
75                  'anchor': 'x',
76                  'showgrid' : False
77              },
78              margin={'l': 40, 'b': 40, 't': 10, 'r': 0},
79              hovermode='closest'
80          )
81      }
```