BUT Science des Données 2 - Semestre 4

COMPETENCE 1 : Traiter des données à des fins décisionnelles

Ressource : **Programmation web (VCOD)**

F.GARNIER

PLAN DE LA RESSOURCE

Projet noté et/ou QCM

Chapitre 2 : Le framework javascript Plotly

1. Introduction

Le framework `plotly` en programmation web est très intéressante, car il a beaucoup d'avantages :

- Nombreux type de graphique
- Interactivé possible
- Disponible aussi dans R et Python (ainsi que d'autres langages)

Pour utiliser plotly en programmation web cela, la première possibilité est de placer le code ci-dessous dans la balise `<head>` de votre fichier HTML :

```
<script src="https://cdn.plot.ly/plotly-2.35.2.min.js" charset="utf-8"></script>
```

Il est bien évidemment possible de télécharger le fichier dans le répertoire de votre page web, et d'y faire référence directement (avec `src="plotly-2.35.2.min.js"` uniquement).

Pour information, cela créé un objet javascript nommé `Plotly` dans la page web.

2. Création d'un graphique en barres

Les fonctions de création de graphique nécessitent le passage en paramètre de **l'identifiant de la balise** dans laquelle placer le graphique. Ainsi, le code JS doit être exécuté après la création de celle-ci.

```
<div id="graphBarres"></div>
```

Ensuite, on va, dans une balise de script, procéder en deux étapes :

1. Créer une variable (nommée `data` ici), contenant les valeurs à mettre en x et en y, et le type de graphique
2. Appeler la fonction `newPlot()` de l'objet `Plotly` avec l'identifiant de la balise où placer le graphique et les données du graphique.

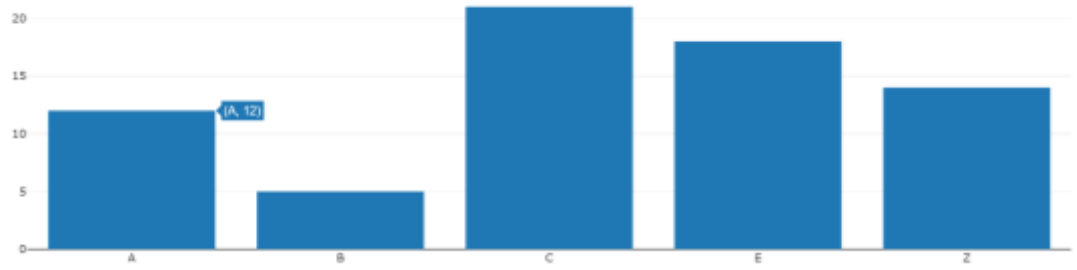
```
<script>

        var valeurs = [12, 5, 21, 18, 14],
            modalites = ["A", "B", "C", "E", "Z"];

        var data = [ {  x:modalites,
                        y:valeurs,
                        type:'bar'
                     }
                   ];

Plotly.newPlot('graphBarres', data);

</script>
```



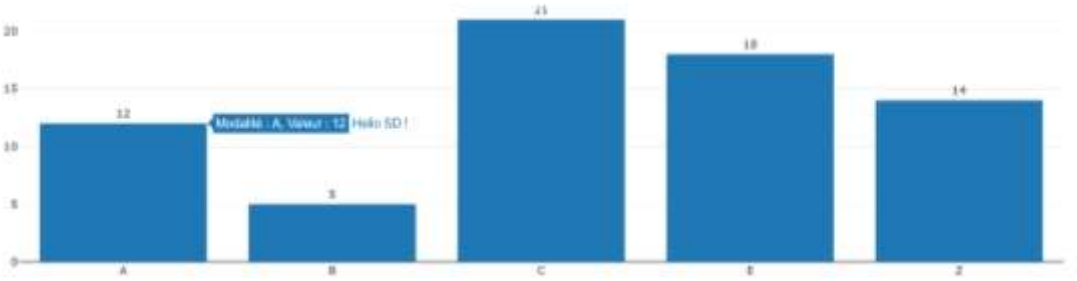
En ajoutant le champ `text` dans les paramètres de la variable `data`, on peut par exemple afficher les valeurs en haut de la barre directement (`text:valeurs`). Si on ajoute en plus le champ `textposition` avec la valeur `"outside"`, le texte est écrit au-dessus des barres.

On peut également customiser la pop-up qui s'affiche en passant la souris sur une barre, avec le champ `hoverinfo` qui peut prendre les valeurs suivantes, entre autres :

- 'none' -> supprime l'affichage
- 'x' ou 'y' -> pour ne voir que l'une des deux informations

On peut aller plus loin avec `hovertemplate` qui permet de redéfinir la façon d'écrire la pop-up, avec utilisation des valeurs des variables en utilisant `%{variable}`. La balise `<extra>` permet d'ajouter des informations.

```
hovertemplate:'Modalité : %{x}, Valeur : %{y}<extra>Hello SD !</extra>'
```

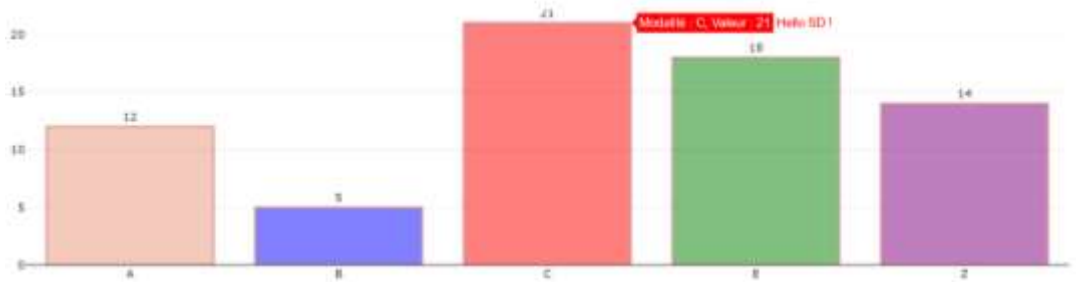


Le champ `marker` permet de changer les couleurs utiliser. Il faut lui passer comme valeur un littéral avec différents champs :

- `color` permet donc de changer la couleur des barres
  - si une seule valeur : même couleur pour toutes les barres
  - si on veut une couleur par barre, il faut un tableau d'autant de couleur
- `line` permet de spécifier une bordure (en indiquant la couleur et la taille éventuellement)
- `opacity` permet de définir une opacité entre 0 (invisible) et 1 (totalement visible)

Exemple :

```
{   color:"DarkSalmon",
    line: { color: "FireBrick",
            width: 2
          },
    opacity:0.5
}
```



Si on change la valeur passée à `color` avec le code ci-dessous, cela permet d'avoir la barre de la modalité B avec une couleur différente :

```
modalites.map(function(e) { return e == "B" ? "DarkRed" : "DarkSalmon" })
```

Remarques :

- la fonction `map()` sur un tableau exécute la fonction (anonyme ici) passée en paramètre à chaque valeur du tableau.
- la fonction passée en paramètre a un ou deux paramètres possibles
  - si un seul : la valeur de l'élément (nommé e ici)
  - si deux : la valeur et la position
- le formalisme `condition ? valeur_si_vrai : valeur_si_faux` en JS permet de faire un test et de renvoyer une valeur spécifique en fonction du résultat du test.

3. Création d'un graphique « nuage de points »

Soit les données suivantes (même *X* pour tout le monde mais *Y* spécifique à chaque modalité):

```
var X = [ 1,  2,  3,  4,  5,  6],
    A = [18, 12, 16,  9, 17, 17],
    B = [ 9,  4,  6,  7,  3,  5],
    C = [ 1,  6,  4,  2,  5,  2],
    E = [15, 14, 10, 12, 13, 16],
    Z = [ 8, 12,  7, 15, 11,  9];
```

Avec la zone pour le graphique suivante :

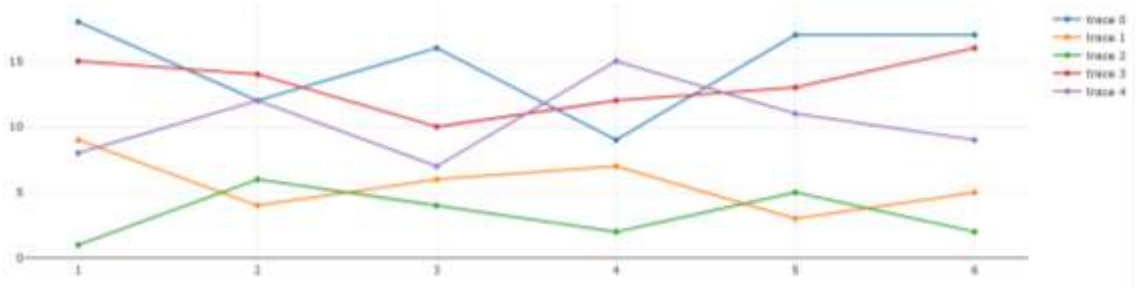
```
<div id="graphNuage"></div>
```

Dans une balise de script, il faut procéder en trois étapes :

1. Créer une variable par modalité (`traceA` pour *A*, …), contenant les valeurs à mettre en `x` et en `y`. *Dans Plotly, on parle de* ***trace ;***
2. Créer une variable (`data`), qui sera une liste contenant les 5 variables créées précédément ;
3. Appeler la fonction `newPlot()` de l'objet `Plotly` avec l'identifiant de la balise où placer le graphique et les données du graphique :

```
var traceA = { x: X, y: A, type: "scatter" },
    traceB = { x: X, y: B, type: "scatter" },
    traceC = { x: X, y: C, type: "scatter" },
    traceE = { x: X, y: E, type: "scatter" },
    traceZ = { x: X, y: Z, type: "scatter" };

var data = [traceA, traceB, traceC, traceE, traceZ];
Plotly.newPlot("graphNuage", data);
```



Comme le type est `"scatter"`, le graphique créé est un ensemble de lignes avec chacune une couleur et identifiées *traceA, traceB*… dans la légende.

Automatisation

L'objet javascript `window`, automatiquement créé lors du chargement d'une page, contient l'ensemble des variables créées dans cette page. Il est donc possible d'accéder à leur valeur avec une code du type `window["A"]` pour l'objet `A`. Ainsi, nous pouvons automatiser la création du graphique précédant, avec le code ci-dessous, en remplaçant la variable `data` par l'appel de la fonction anonyme :

```
Plotly.newPlot("graphNuage", modalites.map(function(e)
                {
                        return { x: X, y: window[e], type: "scatter" }
                })
                );
```

Le champ `name` dans chaque trace permet de changer ce qui est affiché dans la légende. Ici, on doit ajouter `name:e` dans l'objet retourné.

<u>Remarque</u> : les légendes sont cliquables pour faire apparaître/disparaitre les séries de données

Chaque trace est une ligne avec un point à chaque valeur. On peut modifier cela en ajoutant le champ `mode` à l'objet retourné. Celui-ci peut prendre plusieurs valeurs dont les suivantes :

- `"lines"` : uniquement la ligne
- `"markers"` : uniquement les points
- `"lines+markers"` : les deux (idem si on ne met pas le champs)

Lors de l'appel de la fonction `newPlot()`, on peut ajouter des **options dans un troisième paramètre** (souvent nommé `layout` quand on le définit en amont de l'appel). Dans le code ci-dessous, on a fait les opérations suivantes :

- Ajout d'un titre
- Définition de l'étendu de l'axe Y
- Modification de la taille de la police d'écriture de la légende :

```
Plotly.newPlot("graphNuage", modalites.map(function(e) {
          return {
                              x: X,
                              y: window[e],
                              type: "scatter",
                              name: e}
                              }) ,
                    { // 3ème paramètre
                    title: "Evolution des modalités",
                    yaxis: { range: [0, 20] },
                    legend: { font: { size: 20 }}
    });
```

Lorsqu'on passe la souris sur un graphique `Plotly`, un menu apparaît en haut à droite du graphique, avec les éléments suivants :

- Téléchargement du graphique en PNG
- Zoom
- Déplacement dans le graphique (si zoom)
- Sélection par rectangle
- Sélection par *lasso*
- Zoom avant et arrière
- Mise à l'échelle automatique
- Remise à zéro des axes

Celui-ci peut être personnaliser, en ajoutant un quatrième paramètre à la fonction `newPlot()` (si rien à passer en 2ème et 3ème paramètres, mettre un objet vide `{}`), qui peut contenir les champs suivants (entre autres) :

- `displayModeBar` : affiche tout le temps le menu si `true`, et ne l'affiche jamais si `false`
- `staticPlot`: si `true`, rend le graphique totalement statique (et non cliquable)
- `modeBarButtonsToRemove` : qui permet de supprimer certains boutons (sous forme de liste)
  - quelques valeurs : `"pan2d"`, `"select2d"`, `"lasso2d"`, `"resetScale2d"`, `"zoomOut2d"`, `"zoomIn2d"`, `"zoom2d"`…
- `displaylogo` : si `false`, cache le logo Plotly

4. Interaction entre deux graphiques

Il est possible de gérer les évènements sur une page. Pour cela, on créé ce qu'on appelle des *(event) listeners* qui surveille une activité particulière (clic de souris, mouvement de souris, touche appuyée…). Pour plus d'informations, je vous conseille de visiter cette page très complète.

Pour créer un *listener*, il faut préciser sur quel élément nous l'ajoutons, avec la fonction `getElementById()` par exemple. Dans le code ci-dessous, on surveille la balise dans laquelle nous avons placé le graphique en barres.

L'évènement suivi ici (`"plotly_click"`) est de type particulier puisque spécifique à Plotly, mais il correspond à un clic sur un des éléments du graphique.

La fonction *listener* prend donc en paramètre l'évènement suivi et une fonction qui sera exécutée lors de la survenue d'un évènement (ici, donc, un clic sur une des barres).

Cette fonction passée en paramètre (anonyme ici) a un paramètre possible, qui va nous permettre d'accéder aux données de la barre sur laquelle on a cliqué. Dans notre cas, cet objet a un champ `points` qui contient les barres sélectionnés. On ne va prendre que la première, que l'on va stocker dans une variable `infos`.

On créé ensuite ensuite un tableau de couleurs, avec une couleur spécifique pour la barre choisie, grâce au code vu précédemment et le champ `label` de notre variable `infos`.

Ensuite, nous mettons à jour les graphiques à l'aide de la fonction `restyle()` de Plotly qui permet, comme son nom l'indique, de modifier le style d'un graphique.

- Pour les barres, on change donc les couleurs avec la liste créé avant
- Pour les lignes, on fait cela en deux étapes :
    - on change les couleurs de toutes les lignes pour les mettre en gris
    - on change la couleur de celle choisie, connue grâce à la position présente dans le champ `pointIndex` de la variable `infos` :

```
document.getElementById("graphBarres").on(
    'plotly_click',
    function(e) {
        var infos = e.points[0],
            couleurs = modalites.map(function(d) { return d == infos.label
? "DarkRed" : "lightgray" });
        Plotly.restyle("graphBarres", { "marker.color": [couleurs] })
        Plotly.restyle("graphNuage", { "marker.color": "lightgray" })
        Plotly.restyle("graphNuage",   {   "marker.color":   "DarkRed"   },
[infos.pointIndex])
    }
);
```

⇨ **TD2**