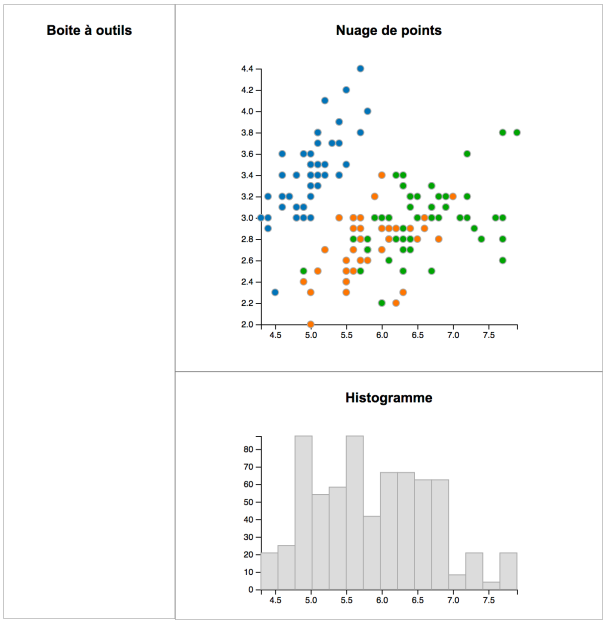
Visualisation d'informations

TP3 : Interaction

Dans ce TP, vous allez apprendre à mettre en place différents modes d'interaction dans des vues coordonnées.

Exercice 1 : Se familiariser avec le code

Téléchargez le dossier exo1.zip et décompressez-le. Ouvrez le fichier interaction.html dans votre navigateur. Vous devriez normalement voir apparaître la visualisation suivante :



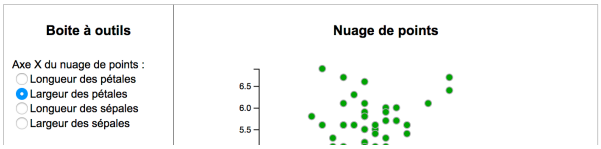
Ouvrez le fichier fleurs.json dans un éditeur de texte. C'est le jeu de données utilisé dans la visualisation. Chaque item correspond à une fleur. Chaque fleur a quatre attributs : une espèce (`species`), une longueur de pétales (`petal_length`), une largeur de pétales (`petal_width`), une longueur de sépales (`sepal_length`) et une largeur de sépale (`sepal_width`).

Ouvrez maintenant le fichier interaction.html dans un éditeur. La page Web contient 3 `divs`, un pour la « boîte à outils », un pour le nuage de points et un pour l'histogramme. Ensuite, un code JavaScript charge les données et appelle deux fonctions : `initCloud` et `initHistogram`. Ces fonctions se trouvent dans les fichiers cloud.js et histogram.js. Ouvrez ces deux fichiers et assurez-vous que vous comprenez le code qu'ils contiennent (toutes les fonctionnalités utilisées ont été vues en TP l'année dernière).

Exercice 2 : Choisir les attributs visualisés à l'aide du nuage de points

Ajoutez dans le `div` de la boite à outils des boutons radios permettant de sélectionner l'attribut à projeter sur l'axe X du nuage de points :

```
<p>
 Axe X du nuage de points :<br />
 <input type="radio" name="radioXCloud">Longueur des pétales<br />
 <input type="radio" name="radioXCloud">Largeur des pétales<br />
 <input type="radio" name="radioXCloud">Longueur des sépales<br />
 <input type="radio" name="radioXCloud">Largeur des sépales
</p>
```



Ajoutez au fichier cloud.js une fonction `updateCloudX(att)` qui affiche, avec un `alert`, la valeur du paramètre `att`.

Ajoutez aux boutons radios un événement onclick qui appelle la fonction `updateCloudX(att)` avec comme valeur pour `att` le nom de l'attribut correspondant au bouton radio. Par exemple :

```
<input type="radio" name="radioXCloud"
onClick="updateCloudX('petal_length')" checked>Longueur des pétales<br
/>
```

Vérifiez que lorsque vous cliquez sur les boutons radios, le nom de l'attribut ainsi sélectionné est affiché.

Supprimez le `alert` de la fonction `updateCloudX(att)` et mettez à jour l'axe X en fonction de l'attribut sélectionné :

```
 attXCloud=att;
 scaleXCloud.domain([d3.min(dataCloud, function(d) { return
d[attXCloud]; }), d3.max(dataCloud, function(d) { return d[attXCloud];
})]);

 gxAxisCloud.call(xAxisCloud);
```

Vérifiez que l'axe se met à jour lorsque vous cliquez sur les boutons radios. Mettez maintenant à jour la position des cercles :

```
items.attr("cx", function(d) {return 25+scaleXCloud(d[attXCloud]);});
```

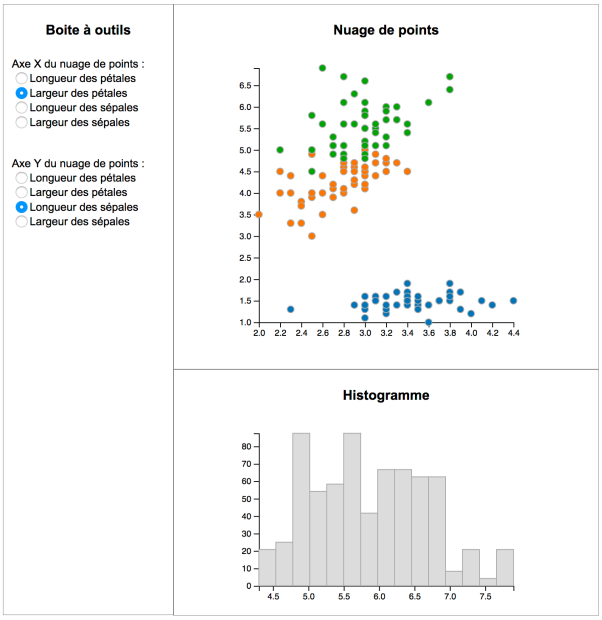
Vérifiez que la position des cercles est mise à jour lorsque vous cliquez sur les boutons radios.

Afin d'animer les transitions entre les différentes positions, appelez la fonction `transition` avant de donner les nouvelles positions :

```
items.transition().attr("cx", function(d) {return
25+scaleXCloud(d[attXCloud]);});
```

Ajoutez aussi une transition sur l'axe X (avant d'appeler la fonction `call`).

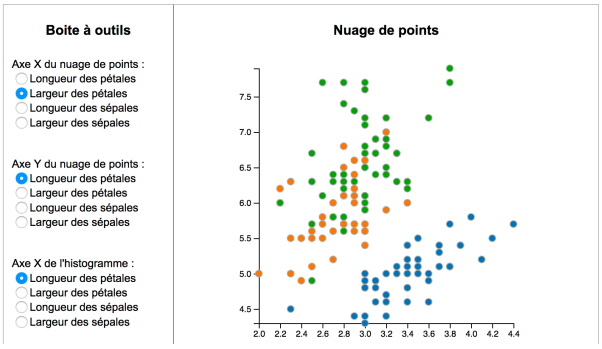
De la même façon, faites en sorte que l'utilisateur puisse choisir l'attribut projeté sur l'axe Y. Vérifiez que tout fonctionne correctement.



Exercice 3 : Choisir l'attribut visualisé à l'aide de l'histogramme

Comme dans l'exercice précédent, ajoutez des boutons radios afin que l'utilisateur puisse sélectionner l'attribut représenté dans l'histogramme. Appelez la fonction `updateHistX(att)` lorsque l'utilisateur clique sur ces boutons.

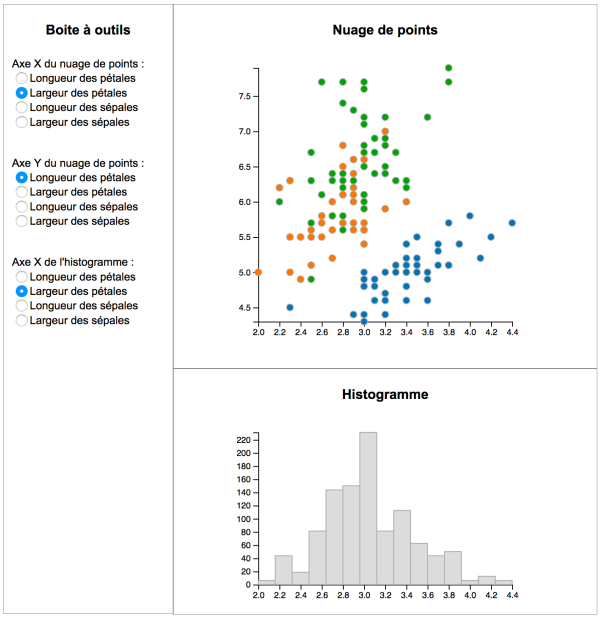
Ajoutez la fonction `updateHistX(att)` à la fin du fichier histogram.js, et mettez-y `alert(att)` pour tester que l'appel fonctionne correctement.



Supprimez le `alert` et mettez à jour l'axe X en fonction du nouvel attribut. Vérifiez que votre code fonctionne correctement.

Mettez à jour l'axe Y. Attention : pour cela, il faut recalculer les densités de chaque classe, en recopiant la boucle de la fonction `initHistogram` qui le fait lors de l'initialisation. Vérifiez que votre code fonctionne correctement.

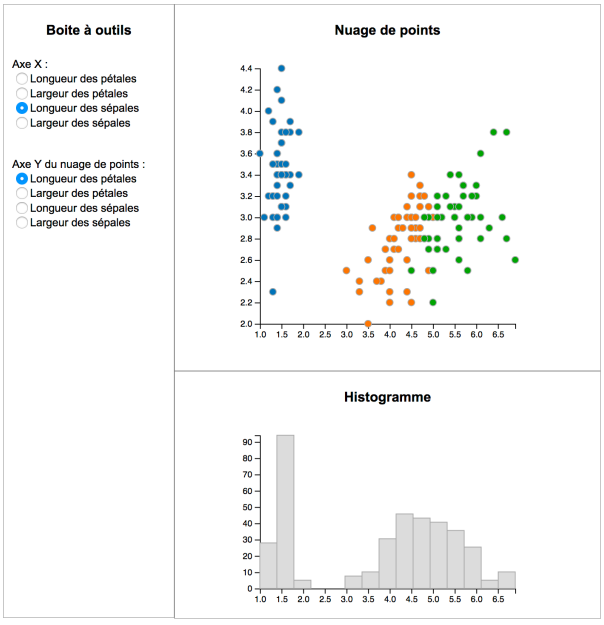
Pour finir, mettez à jour la position y et la hauteur des barres de l'histogramme.



Exercice 4 : Coordonner les axes X des deux vues

Supprimez les boutons radios permettant de modifier l'attribut de l'histogramme et modifiez le texte « Axe X du nuage de points :» en « Axe X :»

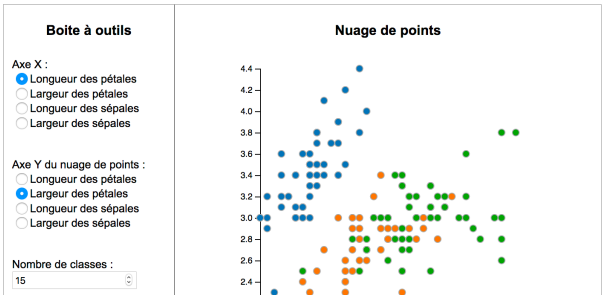
Appelez la fonction `updateHistX(att)` dans la fonction `updateCloudX(att)` pour que l'histogramme soit modifié lorsque l'on change l'attribut en X dans le nuage de points. Les deux vues sont maintenant coordonnées.



Exercice 5 : Modifier le nombre de classes de l'histogramme

Ajoutez dans la boite à outil un `input` de type `number` permettant à l'utilisateur de sélectionner le nombre de classes de son histogramme :

```
<input type="number" id="nbClasses" onchange="updateNbClasses()"
value=15>
```



Le code ci-dessus permet d'appeler une fonction `updateNbClasses()` lorsque l'utilisateur modifie la valeur de l'`input`. Créez cette fonction à la fin du fichier histogram.js et vérifiez qu'elle est bien appelée lorsque vous modifiez la valeur de l'`input`.

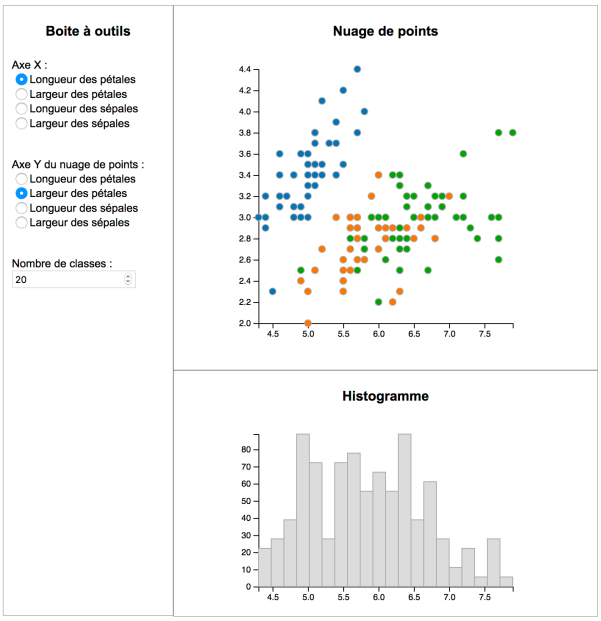
Dans la fonction `updateNbClasses()`, récupérez la valeur que l'utilisateur a saisi dans l'`input`, et donnez cette valeur à la variable `nbClasses` :

```
nbClasses = document.getElementById("nbClasses").value;
```

Vous allez maintenant mettre à jour votre histogramme. Cette fois, comme le nombre de barres a changé, le plus simple est de supprimer tous les objets graphiques de votre svg et de rappeler la fonction `initHistogram` :

```
svgHist.selectAll("*").remove();
initHistogram(svgHist, wHist, hHist, dataHist, attHist);
```

Testez votre application.



Exercice 6

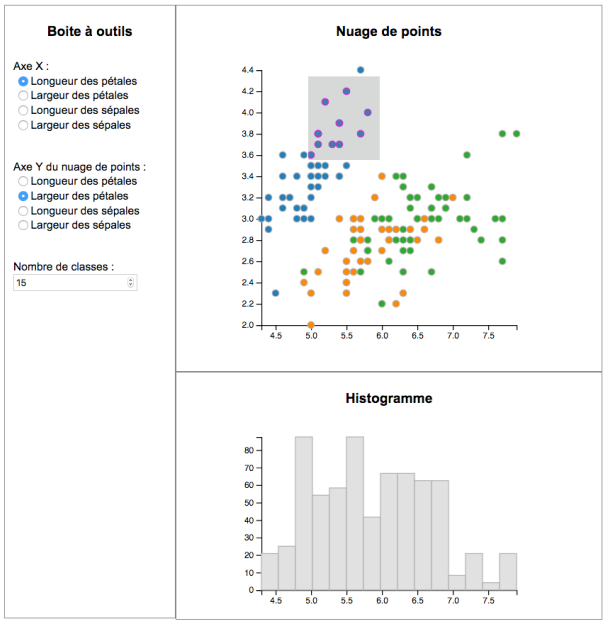
Dans cloud.js, ajoutez un *brush* à votre visualisation (cf. cours de sémiologie graphique). Le code suivant doit être placé à la fin de la fonction d'initialisation :

```
var b = d3.brush();
b.on("brush", function({selection}){
      // Code à exécuter lors de la sélection
});
svgCloud.call(b);
```

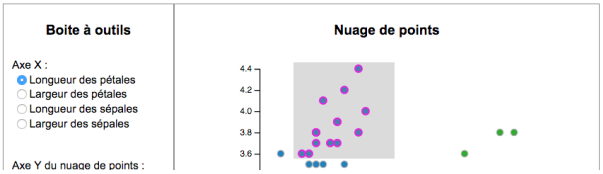
Vous allez maintenant ajouter un code à la fonction afin que les bordures des items sélectionnés apparaissent en magenta. Pour cela, dans la fonction du *brush*, vous allez modifier la valeur de l'attribut `stroke` des items sélectionnés. L'idée est, pour chaque item, de modifier la valeur en magenta si sa position (`d3.select(this).attr("cx"), d3.select(this).attr("cy")`) est contenue dans le rectangle défini par la position en haut à gauche du *brush* (`selection[0][0], selection[0][1]`) et la position en bas à droite du *brush* (`selection[1][0], selection[1][1]`), ou de revenir à la couleur initiale sinon :

```
items.attr("stroke", function(d) {
      var c = d3.select(this);
      if(selection[0][0] <= c.attr("cx") &&
         c.attr("cx") <= selection[1][0] &&
         selection[0][1] <= c.attr("cy") &&
         c.attr("cy") <= selection[1][1]){
            return "magenta";
      }
      else {
            return "#aaaaaa";
      }
});
```

Testez que le changement de couleur s'effectue correctement.



De la même façon, modifiez la largeur des bordures (attribut `"stroke-width"`) des items sélectionnés : 2px au lieu de 1px.

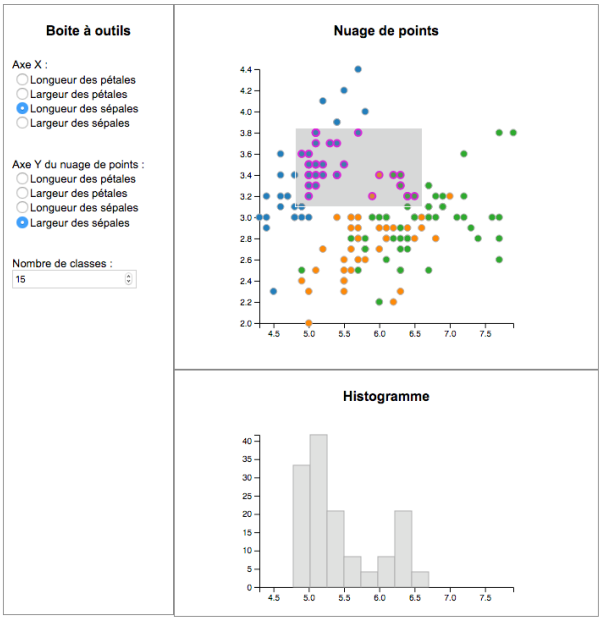


En utilisant le *brush* de votre visualisation, sélectionnez des sommets et modifiez les attributs projetés sur les axes X et Y. Vous pouvez ainsi voir comment ces sommets sélectionnés se répartissent en fonction des attributs sélectionnés.

L'idée maintenant est de mettre à jour l'histogramme en fonction des items sélectionnés. Pour commencer, ajoutez un attribut booléen nommé `selected` aux items juste avant de modifier leur couleur : `d.selected = true;` avant de les mettre en magenta, `d.selected = false;` avant de les mettre en gris. Ensuite, à la fin de la fonction du brush, appelez une fonction `updateWithBrush()`.

Ajouter la fonction updateWithBrush() à la fin de histogram.js. Elle appelle la fonction `updateHistX` en lui passant `attHist` en paramètre.

Modifiez `updateHistX` de façon à mettre à jour votre histogramme en fonction des items sélectionnés (il suffit de modifier la condition d'ajout à la fréquence absolue de façon à ce que seuls les items sélectionnés soient pris en compte (`&& dataHist[j].selected`). Testez votre visualisation et vérifiez que l'histogramme est bien mis à jour.



Un problème se pose alors lorsqu'aucun sommet n'est sélectionné : l'histogramme est vide. Modifiez votre code de façon à ce que l'histogramme soit calculé avec tous les items lorsque aucun d'entre eux n'est sélectionné.

Pour finir, vous pouvez remarquer que lorsque l'on change le nombre de classes, l'histogramme créé n'est plus basé sur l'ensemble des sommets sélectionnés, et pour cause, c'est la fonction `initHistogram` qui le modifie. Adaptez cette fonction afin qu'elle prenne en compte les sommets sélectionnés. Testez votre visualisation.