TP 2 : Vega-Altair

idir.benouaret@epita.fr

Consignes

- Travail à réaliser en binôme
- Bien lire le tutoriel et surtout regarder la documentation officielle : `https://altair-viz.github.io/`
- Pour chaque question/exercice, produisez la/les visualisations qui vous semble être la/les plus appropriée(s).
- Vous pouvez vous inspirer de la galerie des visualisations Vera-Altair : `https://altair-viz.github.io/gallery/index.html`
- Consulter `https://clauswilke.com/dataviz/` pour avoir des suggestions de bonnes façons de faire vos visualisations en fonction de ce que vous cherchez à montrer, et aussi d'éviter des erreurs.
- Il faut produire un jupyter notebook qui va contenir :
  1. les codes ;
  2. les visualisations
  3. et aussi vos commentaires (ce que montre la visualisation)
- Rendu Jeudi 12 Mars 23h42 (peut largement se faire pendant la séance).

Introduction

Vega-Altair est une librairie de visualisation de données, gratuite et open-source. Son API simple, conviviale et cohérente, basée sur la puissante grammaire de Vega-Lite, vous permet de passer moins de temps à écrire du code et plus de temps à explorer vos données.

Exercice I : Getting Started (fondamentaux)

L'objectif de cet exercice est d'apprendre les concepts fondamentaux nécessaires pour créer un graphique Altair de base ; à savoir :

- **Data**, **Marks** et **Encodings** : les trois éléments essentiels d'un graphique Altair.
- **Encoding Types :** Q (quantitative), N (nominal), O (ordinal), T (temporel) qui déterminent la représentation visuelle des encodages.
- **Groupement** et **agrégation** : qui permettent de contrôler certains aspects de la représentation des données dans Altair.

Avec une bonne compréhension de ces éléments essentiels, vous serez en mesure de créer une variété de graphiques dans Altair.

La première des choses est l'installation :

```
pip3 install altair vega_datasets
```

Ensuite, ouvrir un jupyter notebook et importer le package altair.

```
import altair as alt
```

Data : Les données dans Altair sont basées sur le dataframe Pandas. Pour cette section, nous utiliserons le jeu de données sur les voitures déjà disponible à l'aide du package `vega_datasets`

```
from vega_datasets import data
cars = data.cars()
cars.head()
```

Une description du dataset est donnée en Figure 1. Rien de nouveau pour vous, Altair utilise des Pandas Dataframes pour gérer les données :

- chaque ligne représente une observation
- chaque colonne représente une variable

Pour plus d'informations, regardez la documentation

L'objet Chart : Avec les données définies, vous pouvez instancier l'objet fondamental d'Altair, le Chart (graphique). Fondamentalement, un Chart est un objet qui sait comment générer un dictionnaire JSON représentant les données et les encodages de visualisation, qui peuvent être envoyés au notebook et rendus par la bibliothèque JavaScript Vega-Lite. Jetons un coup d'œil à quoi ressemble cette représentation JSON, en utilisant uniquement la première ligne des données :

	Name	Miles_per_Gallon	Cylinders	Displacement	Horsepower	Weight_in_lbs	Acceleration	Year	Origin
0	chevrolet chevelle malibu	18.0	8	307.0	130.0	3504	12.0	1970-01-01	USA
1	buick skylark 320	15.0	8	350.0	165.0	3693	11.5	1970-01-01	USA
2	plymouth satellite	18.0	8	318.0	150.0	3436	11.0	1970-01-01	USA
3	amc rebel sst	16.0	8	304.0	150.0	3433	12.0	1970-01-01	USA
4	ford torino	17.0	8	302.0	140.0	3449	10.5	1970-01-01	USA

FIGURE 1: head du dataset sur les voitures

```
cars1 = cars.iloc[:1]
alt.Chart(cars1).mark_point().to_dict()
```

À ce stade, le graphique comprend une représentation au format JSON du dataframe, le type de marque à utiliser, ainsi que des métadonnées incluses dans chaque sortie de graphique :


```
{'config': {'view': {'continuousWidth': 300, 'continuousHeight': 300}},
'data': {'name': 'data-36a712fbaefa4d20aa0b32e160cfd83a'},
'mark': {'type': 'point'},
'$schema': 'https://vega.github.io/schema/vega-lite/v5.8.0.json',
'datasets': {'data-36a712fbaefa4d20aa0b32e160cfd83a': [{'Name': 'chevrolet chevelle malibu',
'Miles_per_Gallon': 18.0,
'Cylinders': 8,
'Displacement': 307.0,
'Horsepower': 130.0,
'Weight_in_lbs': 3504,
'Acceleration': 12.0,
'Year': '1970-01-01T00:00:00',
'Origin': 'USA'}]}}
```


Les marques :

- `mark_point()` : Un point individuel représentant une observation.
- `mark_bar()` : Un rectangle vertical ou horizontal représentant une catégorie ou une plage de valeurs.
- `mark_line()` : Une ligne reliant plusieurs points, souvent utilisée pour représenter des tendances ou des séries chronologiques.
- `mark_area()` : Une zone remplie entre une ligne et un axe, souvent utilisée pour représenter des données cumulatives ou des valeurs d'intervalles.
- `mark_rect()` : Un rectangle, généralement utilisé pour représenter des cartes thermiques ou des matrices de données.
- `mark_text()` : Un texte, utilisé pour ajouter des étiquettes ou des annotations aux graphiques.

- `mark_tick()` : Une marque courte, telle qu'une ligne ou un point, souvent utilisée pour représenter des repères ou des axes.

La liste complète des marques ainsi que leur description :
`https://altair-viz.github.io/user_guide/marks/index.html`

Les encodings : L'étape suivante consiste à ajouter des canaux d'encodage visuel (ou encodages abrégés) au graphique. Un canal d'encodage spécifie comment une colonne de données doit être associée aux propriétés visuelles de la visualisation. Voici quelques-uns des encodages visuels les plus fréquemment utilisés :

- x : Position horizontale (abscisse)
- y : Position verticale (ordonnée)
- color : Couleur de la marque
- size : Taille de la marque
- shape : forme de la marque
- opacity : degré d'opacité de la marque

Pour la liste complète des encodings disponibles :
`https://altair-viz.github.io/user_guide/encodings/index.html`

Les encodages visuels peuvent être créés avec la méthode `encode()`

Exercice : Maintenant que vous avez bien assimilé les bases, prendre un peu de temps pour générer quelques visualisations.

En particulier, il est suggéré d'essayer plusieurs combinaisons (qui ont du sens) de :

- Marques : `mark_point()`, `mark_line()`, `mark_bar()`, `mark_text()`, `mark_rect()`...
- Colonnes : *'Acceleration', 'Cylinders', 'Displacement', 'Horsepower', 'Miles_per_Gallon', 'Name', 'Origin', 'Weight_in_lbs', 'Year'*
- Encodings : x, y, color, shape, row, column, opacity, text, tooltip

Explorez plusieurs visualisations et essayez d'en apprendre plus sur le dataset. En particulier :

- Répondre aux questions ci-dessous
- Quel encoding est adapté aux valeurs continues, et valeurs quantitatives ?
- Quel encoding est adapté aux valeurs discretes, catégoriques (nominales) ?

Q1 – Quelle est la distribution des poids (weight) des voitures en fonction de l'origine ?

Q2 – Comment varie le poids des voitures en fonction du nombre de cylindres ?

Q3 – Quelle est la relation entre la puissance de la voiture (*horsepower*) et la consommation (*miles per gallon*)

Q4 – Montrer l'évolution de la consommation des voitures au cours du temps.

Exercice II : Binning and Aggregation

Nous avons discuté des données, des marques, des encodages et des types d'encodage. La prochaine pièce essentielle de l'API d'Altair est son approche de la binarisation et de l'agrégation des données.

Une opération des plus utilisées en exploration de données est l'opérateur *group-by*. Brièvement, l'opération consiste à **partitionner** les données en utilisant une condition et **appliquer** une fonction d'agrégation pour chacune des partitions et ensuite **combiner** les données avec les résultats d'agrégation. Un exemple est détaillé sur la figure 2

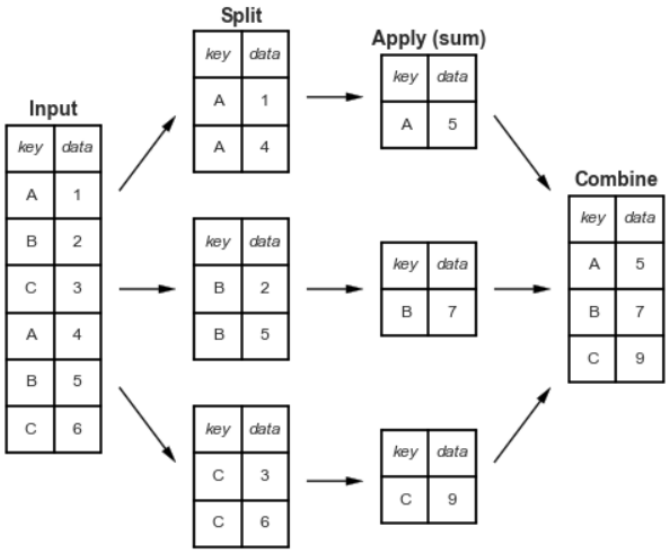

Figure 2: Exemple d'un group-by

Très souvent la visualisation correspondante à une opération de group-by est un histogramme. à noter que pour les données de type catégorique est fait de manière implicite. Par exemple, si on veut afficher la moyenne des *Miles_per_Gallon* en fonction du pays d'origine. Il suffit de faire :

```
alt.Chart(cars).mark_bar().encode(
x='Origin',
y='mean(Miles_per_Gallon)'
)
```

Maintenant, si on veut afficher un histogramme qui doit calculer des bins (variable quantitative) il faut spécifier le paramètre bin= True, dans la fonction alt.X() qui permet de spécifier ce qu'on veut encoder dans l'axe des x.

Par exemple si on devait faire histogramme qui compte le nombre d'enregistrements en fonction de la consommation :

```
alt.Chart(cars).mark_bar().encode(
alt.X("Miles_per_Gallon:Q", bin=True),
y='count()'
)
```

Q1 – Quelle est la répartition de la puissance des voitures par origine ?

Q2 – Comment varie la distribution de l'accélération des voitures en fonction du nombre de cylindres ?

Graphiques composés

Superposer Ici il s'agit de pouvoir avoir la possibilité d'utiliser des marques multiples dans un seul graphique. Un exemple courant est d'avoir à la fois des points reliés par des lignes mais qui représentent les mêmes données.

Regardons à quoi ça ressemble sur les données *stocks*

```
from vega_datasets import data
stocks = data.stocks()
stocks.head()
base = alt.Chart(stocks).encode(
x='date:T',
y='price:Q',
color='symbol:N'
).transform_filter(
alt.datum.symbol == 'GOOG'
)
```

concaténation horizontale On peut aussi concaténer deux charts de manière horizontale, il suffit d'appeler la fonction alt.hconcat() :

```
alt.hconcat(base.mark_line(),
base.mark_circle())
```

à noter qu'il existe aussi un raccourcis pour cette fonctionnalité :

```
base.mark_line() | base.mark_circle()
```

Concaténation verticale Même chose mais en utilisant la fonction `alt.vconcat(chart1, chart2)` ou bien le raccourcis &

```
base.mark_line() & base.mark_circle()
```

Exercice III : Interactivité (Sélections)

RTFM : `https://altair-viz.github.io/user_guide/interactions.html`

L'interactivité d'Altair et sa grammaire de sélection sont parmi ses fonctionnalités les plus importantes. Dans cet exercice, nous passerons en revue les différents types de sélections disponibles et commencerons la réalisation de quelques graphiques interactifs et des dashboards.

- Point Selection : `alt.selection_point()`
- Interval Selection : `alt.selection_interval()`

Interactions basiques : Panning, Zooming, Tooltips Les interactions de base que Altair met à disposition sont le défilement (panning), le zoom et les infobulles (tooltips). Cela peut être réalisé dans votre graphique sans utiliser l'interface de sélection, en utilisant la méthode de raccourci interactive() et le codage des infobulles.

Par exemple, avec notre jeu de données standard sur les voitures, nous pouvons faire ce qui suit :

```
alt.Chart(cars).mark_point().encode(
x="Horsepower:Q",
y="Miles_per_Gallon:Q",
color="Origin",
tooltip="Name"
).interactive()
```

Ce bout de code, très simple et très intuitif à écrire, permet d'afficher une infobulle (tooltip) avec le nom du modèle de voiture (variable :'Name'). La méthode .interactive() permet de pouvoir cliquer/scroller et faire déplacer la visualisation.

Maintenant, on veut ajouter une fonctionnalité de sélection pour se concentrer sur un sous-ensemble de données.

Q1 – Créer un intervalle de selection en utilisant `selection_interval()` et l'ajouter à votre chart : `add_params()`

- `https://altair-viz.github.io/user_guide/generated/api/altair.selection_interval.html#altair.selection_interval`

Vérifiez à l'aide de votre souris qu'on peut sélectionner un sous ensemble de données (région). Cela ajoute une interaction au graphique qui nous permet de sélectionner des points sur le graphique ; l'utilisation la plus courante d'une sélection consiste à mettre en évidence des points en conditionnant leur couleur sur le résultat de la sélection.

Q2 – En utilisant la fonction `alt.condition`, faîtes en sorte que votre graphique affiche en couleur que la zone sélectionnée et affiche en gris les points non sélectionnées

Hint : La fonction alt.condition prend trois arguments : un objet de sélection, une valeur à appliquer aux points dans la sélection et une valeur à appliquer aux points en dehors de la sélection. Ici, nous utilisons alt.value('lightgray') pour nous assurer que la couleur est traitée comme une véritable couleur, plutôt que le nom d'une colonne de données.

Point Selections La fonction `alt.selection_point()` permet à l'utilisateur de cliquer sur des objets de graphique individuels pour les sélectionner, un par un.

Testez les exemples suivants :

```
single = alt.selection_point()

alt.Chart(cars).mark_circle().encode(
x='Horsepower:Q',
y='Miles_per_Gallon:Q',
color=alt.condition(single, 'Origin', alt.value('lightgray'))
).add_params(
single).interactive()
```

La sélection offre une autre option très utile surtout quand la quantité de points (données) est grande. On peut par exemple utiliser `nearest=True` et `on='mouseover'` pour permettre de sélectionner le point le plus proche de la position de la souris. Testez cela :

```
single = alt.selection_point(on='mouseover', nearest=True)

alt.Chart(cars).mark_circle(size=100).encode(
x='Horsepower:Q',
y='Miles_per_Gallon:Q',
color=alt.condition(single, 'Origin', alt.value('lightgray'))
).add_params(
single
).interactive()
```

à noter qu'on peut à l'aide de la touche Shift de votre clavier, sélectionner plusieurs points.

Q3 – Ajouter l'affichage du nom de la voiture ainsi que son Horsepower lorsqu'un point est sélectionné

Interval Selection La fonction `selection_interval()` permet à l'utilisateur de sélectionner un intervalle de valeurs dans son graphique. Par exemple :¨

```
alt.Chart(cars).mark_point().encode(
x='Horsepower:Q',
y='Miles_per_Gallon:Q',
color='Origin:N'
).add_params(
alt.selection_interval()
)
```

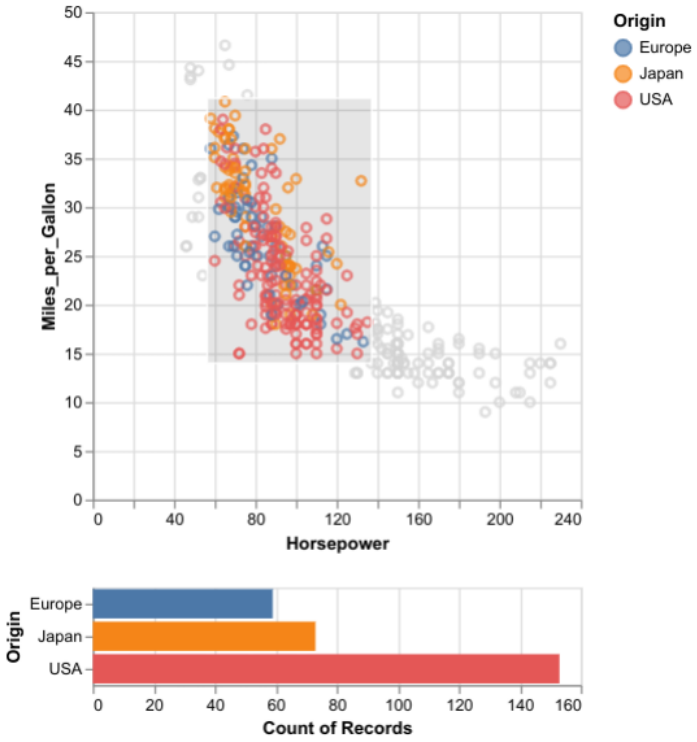

FIGURE 3: Visualisation interactive

Il est aussi possible de créer un intervalle sur seulement l'axe x ou y. essayer le même code si-dessus avec `alt.selection_interval(encodings=['x'])`

Filtrage de données Dans les graphiques à plusieurs charts, il est possible de créer des visualisations dynamiques. c'est à dire utiliser le résultat d'une sélection pour filtrer et n'afficher que les résultats pertinents dans les visualisations.

Q4 – Créer une visualisation avec deux graphiques, un scatter-plot qui montre le nombre de *Miles per Gallon* en fonction du *Horsepower* accompagné d'un histogramme qui compte le nombre d'enregistrements pour chaque pays d'origine. On doit pouvoir sélectionner un intervalle de données qui dynamiquement va mettre à jour l'histogramme en fonction de la zone sélectionnée.

Voici en figure 3 une capture d'écran du résultat attendu

EXERCICE IV : Visualisation sur la dataset des films IMDB

Vous utiliserez le dataset movies, afin de créer quelques visualisations.

```
from vega_datasets import data
movies = data.movies()
movies.head()
```

Warning : les données ne sont pas "propres" : faudra gérer les "Nan", "None", etc

Q1 – Visualiser le nombre de films pour chaque genre

Q2 – Explorer les revenus des films en fonction de leur genre

Q3 – Comment varie les budgets de production au cours du temps ?

Q4 – Visualiser la note des films en fonction de l'année de sortie et utiliser une couleur pour discrétiser par rapport au genre

Q5 – Visualisation qui permet de montrer quels sont les producteurs les plus actifs ? qui ont généré le plus de bénéfices ?

Q6 – Explorer la relation entre la durée des films et les critiques reçues. Est ce que les films les plus longs sont les plus appréciés ?

Q7 – Explorer la relation entre le budget, la note et les revenues générées. Les films à grands budget sont t'ils ceux qui génèrent le plus de revenus ? Sont t'ils les films les plus appréciés ?