Remarque introductive : Je vous rappelle que je n'attends pas de vous que vous fassiez des graphiques jolis et finis. Vous pouvez vous contenter de marqueurs avec des informations temporelles écrites dedans.

Vous avez du commencer à utiliser Folium et maintenant vous souhaitez faire des graphiques à partir de vos données temporelles. Pour faire cela, nous allons utiliser une API appelée VegaLite.

Qu'est-ce qu'est VegaLite ? *https://vega.github.io*
"VegaLite is a declarative format for creating, saving, and sharing visualization designs. With VegaLite, visualizations are described in JSON, and generate interactive views using either HTML5 Canvas or SVG."

Où est la doc ?
https://vega.github.io/vega-lite/tutorials/getting_started.html

Comment savoir si mon json va fonctionner ?
https://vega.github.io/editor/

Un des problèmes de VegaLite est qu'il utilise des fichiers au format JSON pour fonctionner. Or, nous ne sommes pas encore vraiment des pros du format JSON. Nous pourrions utiliser des utilitaires de conversion mais malheureusement, VegaLite demande des fichiers avec des mots-clés particuliers.

Nous allons dans ce tutoriel comprendre ce que l'on doit écrire dans nos variables Python pour que VegaLite sache en faire quelque chose de correct.

Utiliser VegaLite avec Folium

Rien de plus simple, une instruction étant déjà prévu dans Folium :

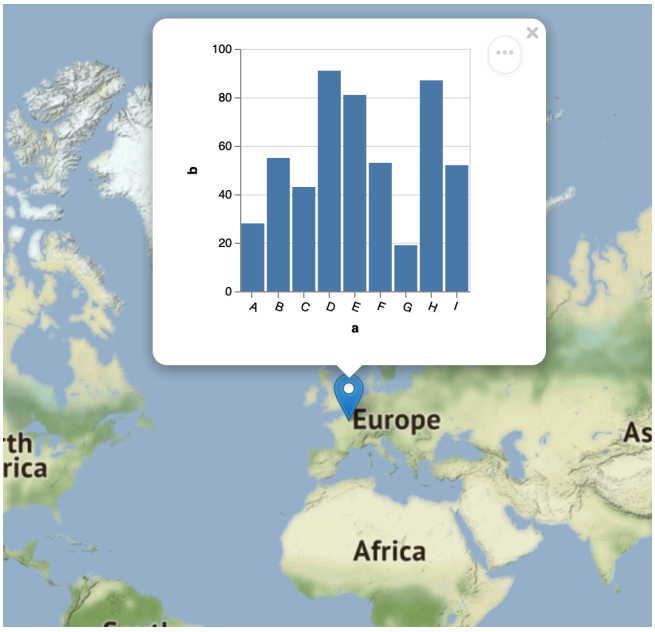
```
startGPS = [48.8, 2.6972]

m = folium.Map(
    location= startGPS,
    zoom_start=2,
    tiles='Stamen Terrain'
)
folium.Marker(
location=startGPS,
popup=folium.Popup().add_child(folium.VegaLite(chart)),
).add_to(m)

m.save('exempleMinimal.html')
```

Évident, il n'y a aucune chance que ce code ne fonctionne car la variable **chart** n'a pas été défini.

Pour définir notre graphique **chart**, nous allons utiliser un dictionnaire de dictionnaires. Chaque entrée du dictionnaire sera un attribut du graphique comme ci-dessous :

2
```
chart ={
  "$schema": "https://vega.github.io/schema/vega-lite/v4.json",
  "description": "A simple bar chart with embedded data.",
  "data": {
    "values": [
      {"a": "A", "b": 28},
      {"a": "B", "b": 55},
      {"a": "C", "b": 43},
      {"a": "D", "b": 91},
      {"a": "E", "b": 81},
      {"a": "F", "b": 53},
      {"a": "G", "b": 19},
      {"a": "H", "b": 87},
      {"a": "I", "b": 52}
    ]
  },
  "mark": "bar",
  "encoding": {
    "x": {"field": "a", "type": "ordinal", "axis": {"labelAngle": 20}},
    "y": {"field": "b", "type": "quantitative"}
  }
}
```

Décrivons chaque entrée.
"**$schema**" : nous envoie sur l'éditeur Vega interactif — optionnel
"**description**" : un bla-bla pour nous aider à savoir ce qu'est ce graphique
"**data**" : dictionnaire contenant plusieurs attributs. L'attribut obligatoire est "**values**". "**values**" est une liste de dictionnaire. Chaque entrée d'un dictionnaire définit la valeur de l'abscisse et la valeur de l'ordonnée.
"mark" : quel type de graphique voulez-vous?
"**encoding**" : dictionnaire contenant plusieurs attributs. Les attributs utiles sont "**x**" et "**y**" qui permettent de définir qui est l'abscisse et qui est l'ordonnée.

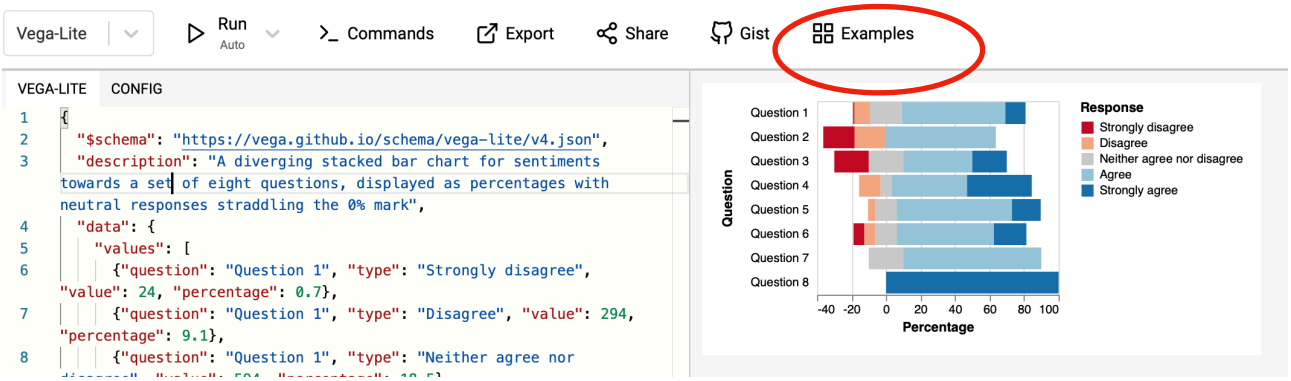


Le résultat de ce code Python est le suivant :

C'est bien mais les diagrammes en barre, ça fait très old-school. Nous, on veut des graphiques avec des courbes qui montent et qui descendent.

Dans ce cas, pas de panique, changez juste `"mark": "bar"` en `"mark": "line"`.

Vous voulez plus d'options ? Je vous conseille d'aller voir dans les exemples de l'éditeur https://vega.github.io/editor/ .



Sauf que si on fait cela, on se retrouve assez vite devant des problèmes de formatage de données (encore…). Il faut vraiment comprendre comment ces valeurs "**values**" doivent être formatées.

Tout d'abord, si je veux des jours plutôt que des catégories, je dois formater mes dates sous une forme très particulière : Année/Mois/Jour avec des slashs. Si je ne le Un petit exemple :

3

```
"data": {
  "values": [
     {"jour": "2020/03/18", "hosp": 31, "rea":2},
     {"jour": "2020/03/19", "hosp": 48, "rea":4},
     {"jour": "2020/03/20", "hosp": 63, "rea":8},
     {"jour": "2020/03/21", "hosp": 71, "rea":16},
     {"jour": "2020/03/22", "hosp": 98, "rea":32},
     {"jour": "2020/03/23", "hosp": 123, "rea":64},
     {"jour": "2020/03/24", "hosp": 160, "rea":128},
     {"jour": "2020/03/25", "hosp": 210, "rea":125}
  ]
}
```

Pour spécifier que ce sont des dates sur l'axe des abscisses, je dois changer le type d'encodage pour l'axe des "**x**". changez juste `"type": "ordinal"` en `"type": "temporal"`.

L'intérêt de VegaLite par rapport à Matplotlib est qu'il nous permet de faire des calculs au vol, en sélectionnant par exemple certaines données, en faisant des moyennes etc. Si vous souhaitez tenter l'aventure, je vous invite à regarder le site internet de VegaLite et à expérimenter sur l'éditeur VegaLite.

Le dernier point qui nous intéresse consiste à tracer plusieurs courbes sur le même graphique. Ca semble en effet pas mal pour visualiser de manière simultanément l'évolution du nombre de patients en réa et celui hospitalisé.

C'est un peu plus complexe mais ça reste jouable grâce à une nouvelle instruction appelée "**repeat**". En gros, cette instruction permet de dire que nous allons avoir plus qu'une donnée à représenter en ordonnée et que chacune de ces données doit avoir des caractéristiques bien à elle.

4

```
{
   "data": {
        "values": [
        {"jour": "2020/03/18", "hosp": 31, "rea":2},
        {"jour": "2020/03/19", "hosp": 48, "rea":4},
        {"jour": "2020/03/20", "hosp": 63, "rea":8},
        {"jour": "2020/03/21", "hosp": 71, "rea":16},
        {"jour": "2020/03/22", "hosp": 98, "rea":32},
        {"jour": "2020/03/23", "hosp": 123, "rea":64},
        {"jour": "2020/03/24", "hosp": 160, "rea":128},
        {"jour": "2020/03/25", "hosp": 210, "rea":125}
        ]
    },
    "repeat": {
        "layer": ["rea", "hosp"]
    },
    "spec": {
        "mark": "line",
        "encoding": {
        "x": {"field": "jour","type": "temporal"},
        "y": {"field": {"repeat": "layer"},"type":
"quantitative","title": "Nombre"},
        "color": {
            "datum": {"repeat": "layer"},
            "type": "nominal"
        }
        }
    }
}
```

C'est bien sympa tout ça mais si vous avez plein de données, comment faire ? La réponse est en vous comme dirait Maitre Yoda.

Plutôt que de passer du côté obscur du copier/coller, nous allons créer une **fonction** qui va prendre comme argument les valeurs "**values**" et va renvoyer les données formatées en JSON prêt à être lu par VegaLite.

Votre fonction va donc principalement être une répétition de ce que nous avons écrit au-dessus, si ce n'est qu'au lieu d'écrire `"values": [ … ]`, vous allez écrire `"values": data` avec data l'argument d'entrée de votre fonction. data sera donc une liste de dictionnaire contenant les données que vous souhaitez représenter…

Tiens donc, cela devrait vous rappeler des trucs et en particulier le formatage que nous avons adopté pour les fichiers CSV… Bref, pour vous proposer un exemple :

5

```
def templateJson(data, titre):
    jsonFile = {
        "data": {
            "values": data
        },
        "repeat": {
            "layer": ["rea", "hosp"]
        },
        "spec": {
            "mark": "line",
            "encoding": {
            "x": {"field": "jour","type": "temporal"},
            "y": {"field": {"repeat": "layer"},"type":
"quantitative", "title": titre},
            "color": {
                "datum": {"repeat": "layer"},
                "type": "nominal"
            }
            }
        }
    }
    return jsonFile
```

Y'a plus qu'à formater correctement vos données et ça va rouler! Par contre, cela ne vous dit pas comment gérer les coordonnées GPS pour afficher vos données au bon endroit ;)