Janvier 2022

Répartition des genres sur Wikipédia

Travail Personnel Encadré

Alexandre Bellebon - 21808613
Maxence Caffier - 21407570
Léo Vincent - 21805239
M2 Informatique - IDC
Année 2022-2023

Table des matières

1 Introduction

1.1 Objectif du travail

L'objectif de ce projet est de représenter la répartition des genres des personnalités présentes dans Wikipédia. On doit donc dans un premier temps trouver un moyen de récupérer nos données. Il existe plusieurs solutions mais chacune d'elles présentes des avantages et des inconvénients (scraping, dump, ...). Il s'agit sans doute de la partie la plus complexe et la plus importante de ce projet. Ensuite, il faut trouver un moyen de représenter ces données et cette répartition pour voir quel filtre il serait intelligent d'implémenter pour obtenir la visualisation la plus pertinente possible. Pour finir, on essaiera de faire un résumé global de nos résultats, mais aussi de porter une analyse critique dessus et de les mettre en relation avec d'autres chiffres.

1.2 Technologies utilisées

Pour réaliser ce projet, on a donc utilisés diverses technologies pour arriver à nos fins.

Pour la récupération des données, on a opté pour python et sa librairie `BeautifulSoup`. C'est une librairie qui permet de parcourir des fichiers HTML. Ceci est donc particulièrement bien adapté au scraping surtout en couplant cela avec la librairie Requests qui a pour but, comme son nom l'indique, de faire une requête à l'URL fourni et de récupérer ainsi le code HTML de la page. On a aussi utilisé la librairie `Pandas` permettant d'extraire des données issues de fichier CSV et de les mettre sous forme de matrice. Enfin, les données récoltées seront converties au format JSON à l'aide de la librairie `json` de python.

Pour la visualisation des ces données, on est parti vers une application web en `Reactjs`. Le framework permettra d'offrir un graphique qui se mettra à jour automatiquement en fonction des filtres que l'utilisateur choisit. Pour le graphique, on s'est servis de la librairie `d3.js` (ainsi que la librairie `Bertin`, qui est spécialisée dans les visualisations de données sur des globes terrestres et des planisphères). C'est une librairie offrant de nombreux graphique personnalisable et facile à intégrer dans une application web. De plus, il existe des modules `d3` pour `React` que l'on peut installer via `npm`, un gestionnaire de paquets. Pour habiller notre application, on a pris la librairie `Grommet` qui possède également un module dans `React`. Et évidement pour faire fonctionner notre site, il y aura dans notre code du HTML, du CSS et du JavaScript (étant donné que `React` est une librairie js).

2 Récupération des données

2.1 Options disponibles

Pour récupérer les données de Wikipédia, il existe diverses solutions énumérées sur le site Wikidata. Il existe également des sites extérieurs permettant de récupérer des données de Wikipédia (ex : YAGO), mais ne comprenant pas leur fonctionnement, on les a laissés de côté. Au final, 3 options ont été retenues :

La première est de récupérer des `dumps` de Wikipédia. Ainsi, on aura toutes les données que l'on souhaite. On pourra alors analyser rapidement une bonne partie des données et sélectionner les pages qui nous intéresse et celle qui ne nous intéresse pas. Cependant le gros inconvénients à cette option, c'est la taille des fichiers. Récupérer la totalité des personnalités de la plateforme représente plusieurs centaines de gigas et serait impossible à traiter pour nous. De plus, via les `dumps`, il n'est pas possible de filtrer ce que l'on récupère, on aurait pu avoir en plus d'autres données (entreprise, personnage fictif, film, ...). L'option parfaite aurait été de pouvoir filtrer ces informations au préalable. Certains sites le propose mais les données sont beaucoup trop spécifiques et pas assez représentatives pour être analysé.

La seconde option est celle de faire du scraping. Elle consiste de parcourir Wikipédia à partir d'une page de départ. De ce fait, en naviguant entre les liens, on capte le maximum d'information concernant la personne (nom, sexe, nationalité, date de naissance) et on les enregistre. Ainsi, on filtre les données permettant donc de générer un fichier moins lourd. Mais il existe des inconvénients à cette stratégie. Tout d'abord, elle est lente puisque l'on doit faire des requêtes pour chaque individu. Et ensuite, garder les informations pertinentes n'est pas simple étant donné que chaque page Wikipédia est différente d'une autre dans sa structure. On était partis sur cette voie au départ, puis finalement on a opté pour la troisième proposition que voici.

La troisième option consiste à récupérer nos informations via une banque de données. Après de multiples recherches, on s'est arrêtés sur le site Kaggle. C'est une plateforme qui offre une grande quantité de dataset dans divers domaines, et notamment quelques-unes sur Wikipédia. Ici en l'occurrence, on a trouvés des données sur les personnalités décédés présentes dans l'encyclopédie, tous langages confondus (vous pouvez y accéder ici). Cette option est à la fois assez simple et permet d'avoir accès à des donnés correctes pour la plupart, même si certaines erreurs peuvent être présentes puisque les données ont été générées grâce à du machine learning. On opta donc pour cette option, même si on aurait préféré créer notre propre base de données.

2.2 Stratégies mis en place

2.2.1 Script de scraping

Même si on a finalement utilisé le dataset issu de Kaggle, on va vous décrire le fonctionnement de notre script de scraping car on a commencés à travailler dessus avant de trouver une autre solution.

Ce script a pour but :

— Envoyer une requête à une adresse pour récupérer le code HTML de la page
— Récupérer (dans la partie droite de la page) la nationalité de la personne
— Récupérer le genre de la personne via sa description
— Enregistrer les informations gagnées
— De détecter les autres liens pour savoir où se diriger par la suite.

La première partie est assez simple. Il suffit d'importer la librairie Request pour, comme son nom l'indique, envoyer une requête à l'adresse donnée et ensuite de donner la réponse que l'on a eu à BeautifulSoup, qui va permettre de récupérer les informations de cette page de manière efficace.

```
response = requests.get("https://en.wikipedia.org"+url)
soup = BeautifulSoup(response.text, "html.parser")
```

Ensuite, la première information que l'on essaye de capter c'est la naissance de l'individu. Car non seulement on peut connaître la nationalité d'une personne par son lieu de naissance, mais en plus si jamais on arrive sur une page qui ne possède pas de champ "Born" (ou "Place of birth" pour les sportifs), ceci indique que l'on est sur une page qui ne concerne pas une personne (Chanson, film, entreprise, ...). S'il y a un bien un champ "Born", alors on récupère la dernier information (donc le pays) et on vérifie s'il n'y a pas de champ "Nationality" ou "Origin". Au cas où, cela indique que la personne n'a pas la même nationalité que son pays de naissance, donc on enregistre cette information à la place de l'ancienne.

```
for label in infobox_labels:
    if label.get_text() in ["Born","Place of birth"] :
        for child in label.parent.findChildren("td"):
            country = child.get_text().split(",")
            nationality = country[len(country)-1]
    elif label.get_text() in ["Origin","Nationality"] :
        for child in label.parent.findChildren("td"):
            nationality = child.get_text()
        break

if nationality == "":
    print(name)
    return 0
```

L'étape suivante est donc de savoir s'il s'agit d'un homme ou d'une femme. Pour se faire, on consulte en premier lieu les catégories auxquels la personnalité est associée. Si une d'entre elles

contient le mot "female" ou "male", alors on lui associe le sexe correspondant. Sinon il reste sa biographie. Pour éviter de perdre trop de temps, on consulte uniquement les deux premiers paragraphes de la page et on consulte la présence ou non de "she/her" pour une femme, et de "he/his" pour un homme. Si cela n'est pas suffisant, alors on ignore la page et on passe à la suivante.

```
for cat in catlinks:
    txt = cat.get_text().lower()
    if " female " in txt:
        gender = "Femme"
    elif " male " in txt:
        gender = "Homme"

if(gender == "") :
    abstract = soup.find_all("div", id="mw-content-text")#.findChildren("
    for ab in abstract :
        for i in range(2):
            txt = ab.findChildren("p", class_="")[i].get_text().lower()
            if " she " in txt or " her " in txt:
                gender = "Femme"
            elif " he " in txt or " his " in txt:
                gender = "Homme"
```

une d'en

Avant de changer de page, on enregistre l'individu avec dans un dictionnaire que l'on réagencera plus tard à l'aide de la librairie JSON de python. Une fois un certain nombre de page visités, on inscrira le tout dans un fichier JSON.

Et en dernier lieu, on cherche tous les liens cliquables présents sur la page. On filtre au maximum pour gagner du temps :

— on évite les catégories ou page hors-sujet avec des " ;"
— on repasse pas par des liens où on est déjà allé
— on vérifie bien qu'on se dirige vers une page Wikipédia

Et on sélectionne un lien aléatoirement parmi tous ceux qui reste.

```
        links = soup.find_all("a")
        for link in links:
            link_url = link.get("href")
            if link_url is not None \
                and link_url.startswith("/wiki/") \
                and not ":" in link_url \
                and link_url not in list_url:
                list_url.append(link_url)
                CrawlingPage(link_url,depth-1)
```

La proportion d'analyse réussi de ce script est assez élevé, cependant sa lenteur est son gros défaut et nécessite de tourner un bon bout de temps pour avoir une base de donnée solide. On a donc décidé de le laisser à l'étape de prototype.

2.2.2 Téléchargement via banque de données

Le jeu de données que l'on a téléchargé au format CSV compte plus de 1.22M d'individus (toutes versions de Wikipédia confondus, extrait en été 2022), soit bien plus que l'on aurait pu récolter via notre script python, expliqué précédemment. Ces données comporte plusieurs champ (Nom, Genre, Date de naissance/décès, pays, métier, âge du décès et cause de la mort) avec certains champ vide.

```
countryList = pc.countries
df = pd.read_csv('externDataset.csv')
df.drop(["Name", "Death year", "Short description", "Manner of death", "Age of death"], axis = 1, inplace = True)
df = df.dropna(subset=['Country', 'Birth year', "Gender"])
```

On a utilisé python et sa librairie `pandas` permettant de convertir les données en matrice (de type `DataFrame`) et ensuite de garder les informations qui intéressante. Dans un l'exemple ci-dessus, on se sert de `pycountry` pour avoir une liste de pays, pour faire plus facilement des recherches dans le flux de données que l'on a, et ensuite on garde uniquement les lignes où les champs "Country","Birth year" et "Gender" ne sont pas nuls.

Ensuite on enregistre les données pertinentes dans des dictionnaires. On en a un total de 3 par pays :

— dicoOverall : Répartition des genres de manière général dans le pays
— dicoPolitician : Répartition homme/femme en politique dans le pays
— dicoArtist : Répartition homme/femme dans le domaine artistique dans le pays

Et ensuite, chaque dictionnaire contient ces informations :

— id : Diminutif du pays en 3 lettres (permettant au graphique de reconnaitre le pays)
— name : Nom du pays à l'affichage
— "0","100",... : Répartition des sexes par siècle + siècles précédents. Les données sont rangées de cette manière `[homme, femme, transgenre, non-binaire]` (transgenre et non-binaire ne sont pas présent pour les dictionnaires politique et artiste)

```
century = info[info['Birth year'] <= i]

dicoPolitician[str(i)] = politicianStat(century)
dicoArtist[str(i)] = artistStat(century)

gender = century.Gender.tolist()
dicoOverall[str(i)] = [gender.count("Male"),gender.count("Female"),gender.count("Transgender Female") + gender.count("Transgender Male"),gender.count("Non-Binary")]
```

On filtre les personnages nés avant l'année i et on transforme tous le contenu du champ "Gender" en liste. Ainsi, avec la fonction `count`, on peut récupérer le nombre d'individu de chacun des genres et le mettre dans un tableau. On fait la même chose pour les dictionnaires particuliers, où là, on filtre, en plus de l'année, avec le champ "Occupation" (dans `politicianStat` et `artistStat`).

Pour certains pays, on est obligés de faire des requêtes personnalisées. C'est-à-dire, dans certains cas, on a des données sur des pays qui n'existent plus, ou alors qui ont un nom différent de celui de `pycountry`. Voici un exemple :

```
case "RUS":
    return df[(df["Country"].str.contains("Russia")) | (df["Country"].str.contains("Soviet"))]
```

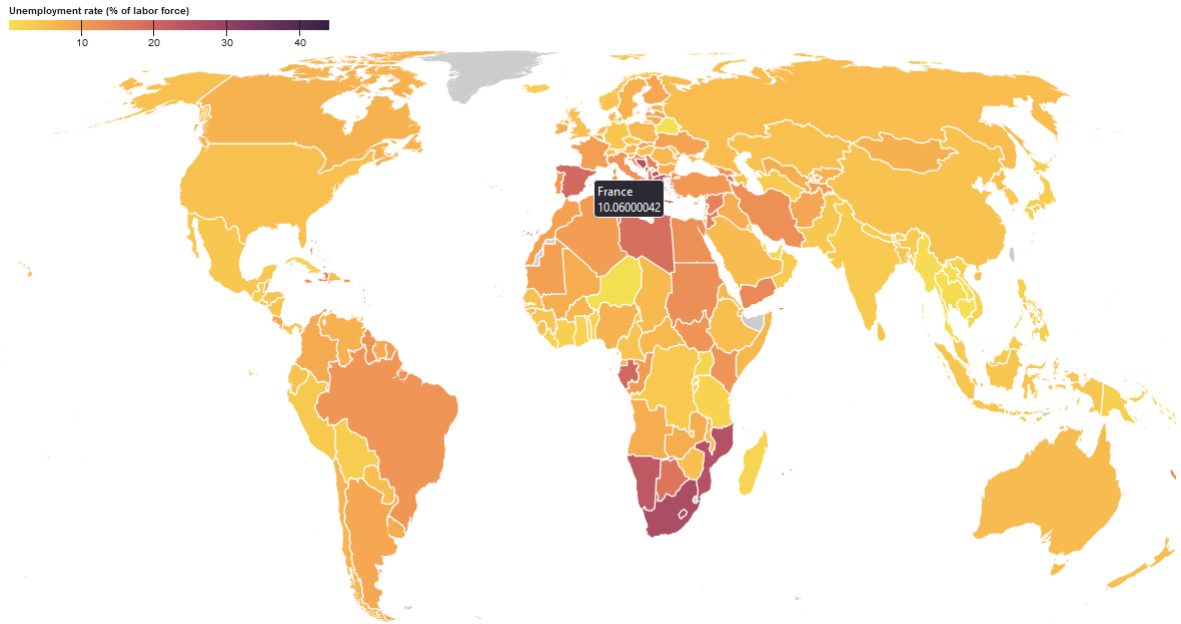
Ici, quand on tombe sur la Russie, on fait à la fois une recherche car on a un nom différent entre la base de données ("Russia") et la librairie `pycountry` ("Russia, Federation of"), et aussi car le pays a changé de nom au cours du temps ("Soviet Union").

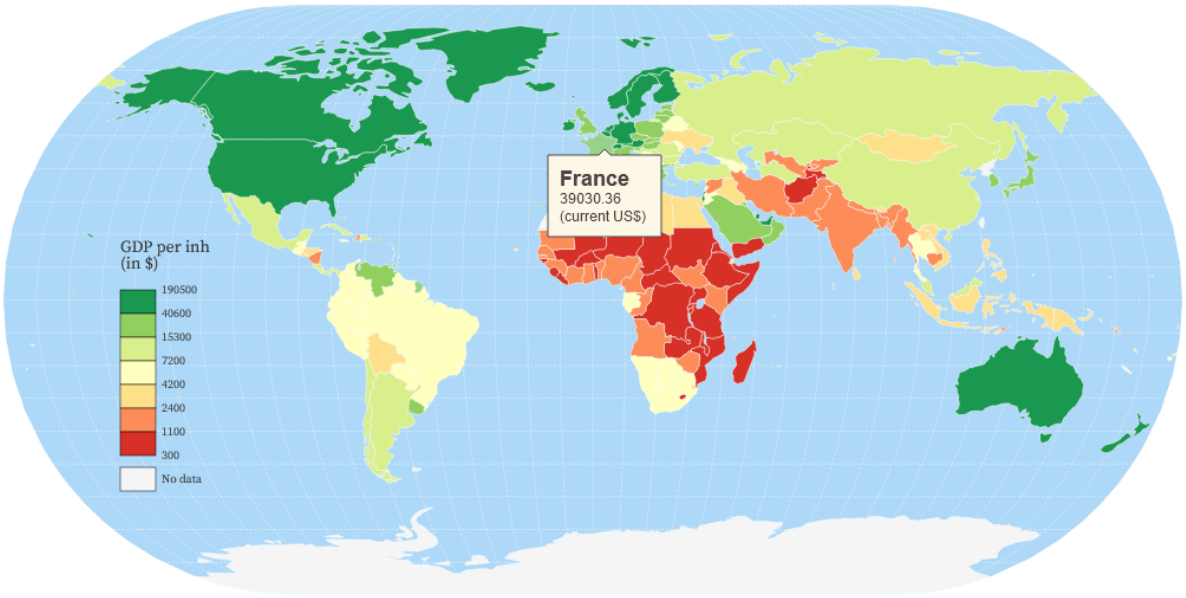
Et pour terminer, on inscrit chacun des dictionnaires dans leur tableau respectif (que l'on a nommé `jsonOverall`, `jsonPolitician` et `jsonArtist`) et une fois les 249 pays de `pycountry` parcourus, on les rentre dans les fichiers `overallData.json`, `politicianData.json` et `artistData.json`. Juste avant, on appelle la fonction `json.dumps` pour formater nos données au format JSON et les indenter automatiquement.

3 Visualisation des données

3.1 Choix du graphique et des filtres

Pour choisir le graphique, on s'est servi de la librairie `d3.js` qui offre un nombre conséquent de modèle de visualisation. L'objectif était de trouver une carte du monde, pour pouvoir illustrer la répartition des genres dans chaque pays où on a des informations. On a retenu deux modèles :





Les deux planisphères sont assez similaires, mais on a sélectionné le second modèle. Son avantage est sa personnalisation. En effet, il utilise la librairie `bertin`, qui est une librairie facile à personnaliser (tooltip, légende, couleur, ...). Le seul inconvénient est justement cette librairie qui doit être incorporer au site. Quand bien même il existe des modules `React` pour `bertin`, sa mise en place n'est pas simple, et en plus de cela elle nécessite une remise en forme des données de notre fichier JSON pour que la librairie puisse les lire.

Ensuite on a réfléchi quels filtres il serait intéressant de mettre en place pour mieux visualiser les données. Le premier est d'établir une frise chronologique permettant d'afficher la représentation des sexes en fonction de l'année de naissances des personnalités. Le second est de pouvoir afficher, à sa guise, soit la proportion d'homme, soit la proportion de femme par pays. Le troisième est de sélectionner un métier (artiste ou politicien), et d'observer l'équilibre dans chaque nation par rapport à ce métier.

On a également prévu d'autres filtres mais par manque de temps, on les a laissé de côtés. Cependant, si vous le souhaitez, on les a énuméré dans la partie 6.2.

3.2 Développement du site réactif

Pour pouvoir offrir la meilleure expérience utilisateur, on a décidé de réaliser une application web en `React`. Ce framework permet, à chaque modification du DOM ou de variable définis, d'appeler des fonctions de mise à jour (`useEffect` et `useState`).

Pour mettre en place le graphique, on utilise donc les modules `d3-react` et `bertin` à l'aide de `npm`. En premier lieu, on use d3 pour lire le fichier `world.geojson.txt`, qui va permettre de pouvoir générer la carte du monde. C'est la seule fois que l'on a appelé nous-même, mais ensuite il sera utilisé indirectement par `bertin`, pour créer notre modèle au format `svg`. Une

fois le fichier lu, on envoie les données de la carte et nos données au format JSON à la fonction `drawChart` (que l'on a détaillé par la suite). Une fois le `svg` terminé, on l'ajoute à un élément `ref` (via l'élément `useRef` de `react`) qui permet d'ajouter le graphique sur la page du site, ou alors de le mettre à jour si l'utilisateur à sélectionné des critères particuliers. Tout ceci se trouve dans la fonction `useEffect` de `react`, qui est appelé, dans cette situation, à chaque modification d'un des filtres.

```
d3.json(jsn).then((json) => {
    let chart
    if(!extra){
      chart = drawChart(json,dataViz);
    }
    else
      chart = drawChartExtra(json,dataViz)
    if(svg.current.firstChild == null)
      svg.current.appendChild(chart)
    else{
      svg.current.removeChild(document.querySelector("svg"))
      svg.current.appendChild(chart)
    }
```

Ensuite pour la fonction `drawChart`, elle consiste à appeler la fonction de `draw` de `bertin` en précisant les critères que l'on veut afficher, la nuance de couleur à utiliser, la légende, les informations présentes dans les tooltips, etc.

```
bertin.draw({
  params: { projection: geoEckert3() },
  layers: [
    {
      type: "layer",
      geojson: bertin.merge(world, "ISO3", data, "id"),
      fill: {
        type: "choro",
        values: "meanGender",
        nbreaks: 5,
        method: "geometric",
        colors: "YlGnBu",
        leg_round: 0,
        leg_title: legend,
        leg_x: 100,
        leg_y: 200
      },
      tooltip: {
        fields: [
          "$name",
          " ",
          (d) => {return(d.properties.meanGender === "" ? "No data collected for this country" : "Men coverage : " + Math.abs(menCoverage - d.properties.meanGender).toFixed(2) + "%")},
          (d) => {return(d.properties.meanGender === "" ? " " : "Women coverage : " + (Math.abs(womenCoverage - d.properties.meanGender).toFixed(2) + "%"))},
          " ",
          (d) => {return(d.properties.meanGender === "" ? " " : "------------ Details ------------")},
          " ",
          (d) => {return(d.properties.meanGender === "" ? " " : "Men : " + d.properties[birthDate][0])},
          (d) => {return(d.properties.meanGender === "" ? " " : "Women : " + d.properties[birthDate][1])}
        ],
        fill: "#add8f7",
        fontSize:[20,15,15,15,15,15,15,15,15],
        col: ["black", "black", "#213f77", "#be34b6","black","black","black","#213f77", "#be34b6"],
        fillOpacity: 0.7
    }
    },
    { type: "graticule" },
    { type: "outline" }
  ]
```

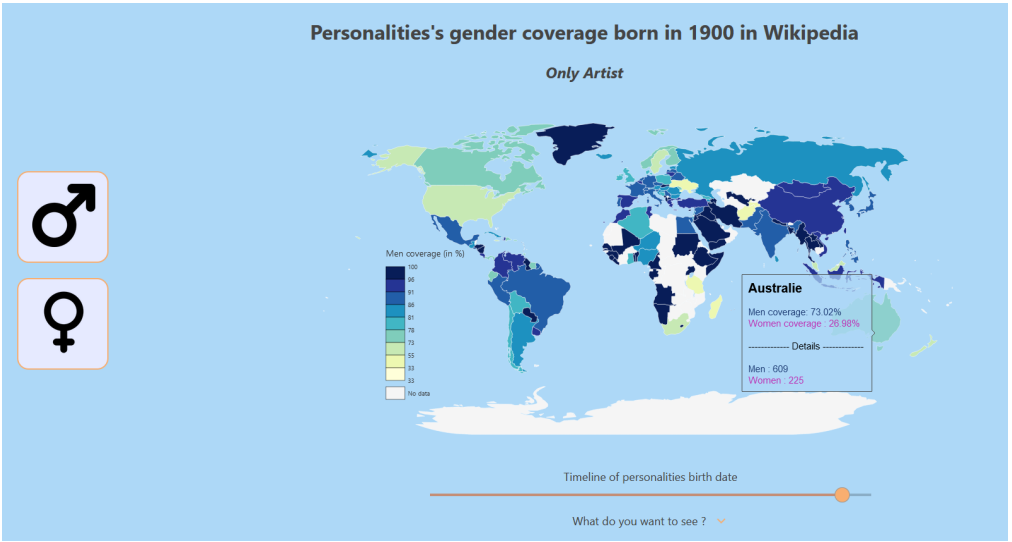
Pour les filtres, on a donc mis en place des `Button`, un `Tip`, un `RangeInput` et un `Menu` issus de `Grommet`. En voici un exemple :

```
<Button
  style={{width:145, height:145, marginTop:100, backgroundColor:"#E6EAFF"}}
  margin="small"
  label={<img src={male} alt="Symbole Masculin" />}
  onClick={() => setReverseViz(false)}
/>
<Button
  style={{width:145, height:145, backgroundColor:"#E6EAFF"}}
  margin="small"
  label={<img src={female} alt="Symbole féminin" />}
  onClick={() => setReverseViz(true)}
/>
```

```
<Tip
  plain
  content={birthDate}
>
  <RangeInput
    color="#C48E76"
    style={{width:700}}
    value={birthDate}
    min={0}
    max={2000}
    step={100}
    onChange={(e) => setBirthDate(e.target.value)}
  />
</Tip>
</div>
<div class="divUnderMap">
  <Menu
    label="What do you want to see ?"
    items={[
      { label: 'All', onClick: () => {setOccupation("All")} },
      { label: 'Artist', onClick: () => {setOccupation("Artist")} },
      { label: 'Politician', onClick: () => {setOccupation("Politician")} },
    ]}
  />
```

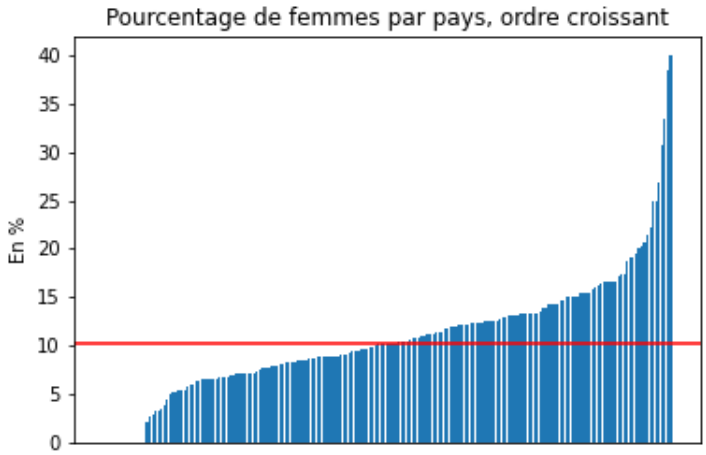
A gauche, on a les boutons. Ces derniers permettent de passer d'un mode de visualisation à un autre à l'aide de la variable `reverseViz` qui change quand on clique sur un des boutons à l'aide de la méthode `useState`, propre à `React` (ici `setReverseViz`). A droite, le range permet quant à lui de modifier le graphique en fonction de l'année de naissance des personnalités. Par exemple, si l'utilisateur sélectionne 1800, il verra la répartition des genres de toutes les personnalités nées avant 1800. Comme pour `reverseViz`, on utilise `useState` (`setBirthDate`) pour mettre à jour la variable `birthDate`. On lui a assorti un `Tip`, permettant à l'utilisateur de se rendre plus facilement compte de la date qu'il sélectionne avec le `RangeInput`. En dessous, on retrouve le dernier filtre, celui associé à la profession des individus. C'est un menu déroulant avec la possibilité de sélectionner spécifiquement les individus catégorisés comme politicien ou artiste (ou alors de voir toutes les professions confondues : athlète,artiste,chercheur,etc.). A l'instar des autres filtres, on se sert `setOccupation` pour mettre la variable `occupation` à jour.

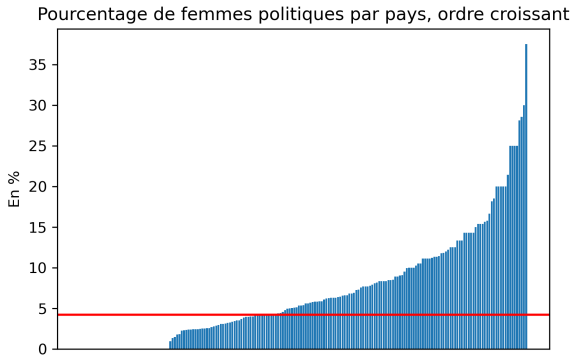
Quand ces variables (`birthDate`, `reverseViz`, `occupation`) sont modifiées via `useState`, la fonction `useEffect` est appelé et actualisera la vue, créant de ce fait un nouveau graphique avec de nouveaux paramètres (et supprimera l'ancien). Voici le rendu de notre application :

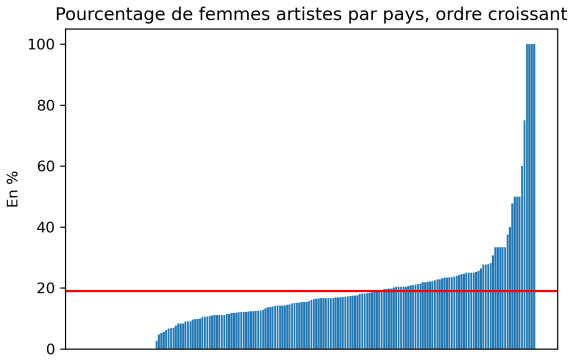


4 Analyse des résultats

Voici quelques graphiques résumant la répartition homme-femme sur Wikipédia :

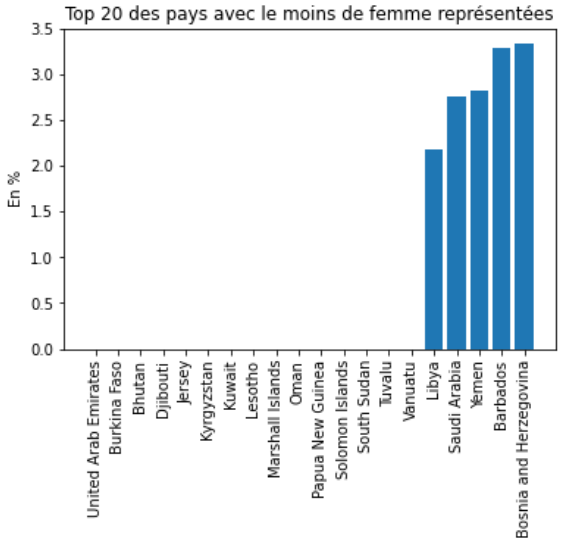


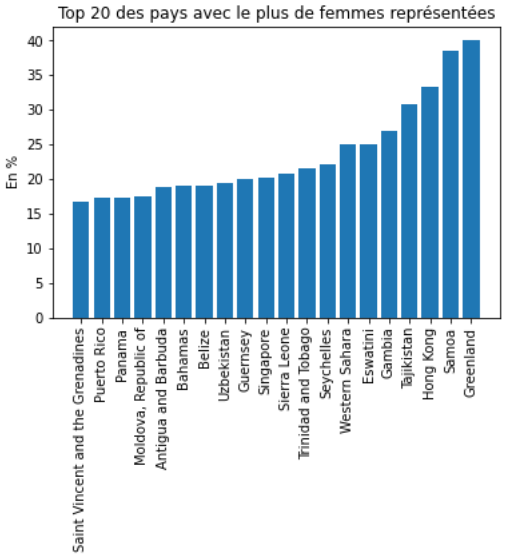




Cela nous montre une grande disparité entre les pays. Certains sont à 0% et d'autres à 40%. Même si tous les pays n'ont pas le même nombre de personnalités, cela reste intéressant. On retrouve donc une moyenne de femme à 10% (10.31%) à l'échelle globale. Concernant les métiers, on note une nette différence entre le monde politique et le monde artistique puisque l'on passe de moins de 5% de femme en politique (4,19%) à pas loin de 20% pour les artistes (19,01%), tous pays confondus. On note même des pays où 100% des artistes sont des femmes. D'ailleurs, on remarque qu'il y a beaucoup plus de pays avec plus de 5% de femme en politique qu'en dessous. Cela signifie que la majeure partie des pays en dessous de cette moyenne sont des pays avec avec beaucoup de données.

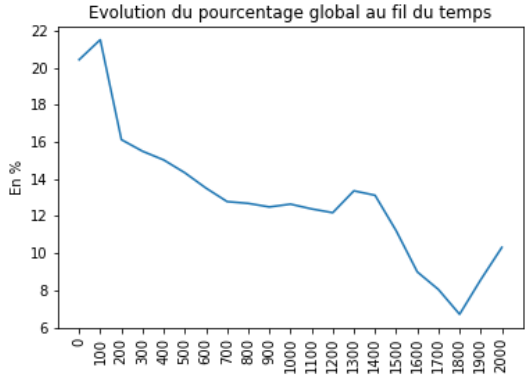
Pour aller un peu plus loin, on peut regarder les pays avec la plus faible proportion de femme et ceux avec la proportion la plus élevé.

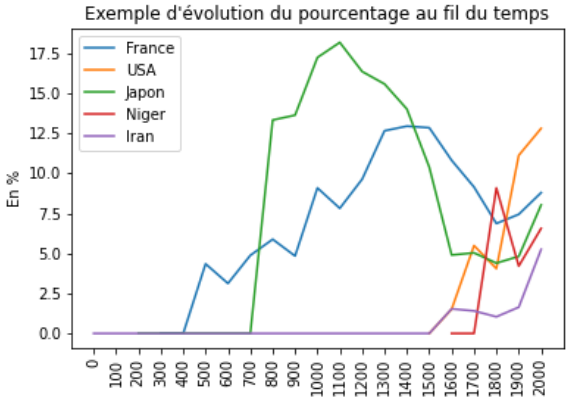




On observe que ces pays/région sont, pour la majorité, peu connus et, logiquement, des pays pour lesquelles on a peu de données. On ne peut mettre en corrélation ce manque d'information avec la région de ces pays étant donné qu'ils viennent des 4 coins du monde. Ce manque de précision amène à des valeurs extrêmes et conclue qu'il est plus pertinent de retenir la moyenne mondiale qui est, de ce fait, assez proche des pays avec beaucoup d'élément. Par exemple, on peut cité les statistique du Brésil (10.29% de femme en générale, 20.40% dans l'art et 2.67% en politique) qui sont très proche des moyennes mondiales.

Mais qu'en est-il de cette équilibre au cours du temps ?





Curieusement, la proportion de femme la plus élevée se trouve bien avant notre époque, aux alentours du Ier et IIème siècle. Cela est du à la présence de certaines reines et impératrices qui, au fur et à mesure des époques, disparaissent pour laisser place de plus en plus la place aux hommes. Cependant, on note une rehausse de la proportion féminine au XIXème siècle, conséquence des premiers combats pour l'égalité homme-femme survenu à la fin du XVIIIème siècle un peu partout en Europe mais aussi dans d'autres région du monde. On retrouve cela dans le graphe suivant qui permettent de comparer l'évolution de certains pays entre eux. Ces pays viennent de diverses régions du globe mais ont malgré tout une évolution assez similaire. Les siècles pré XVIIIème sont assez disparates (France, Japon), on note la même augmentation aux XVIIIème et XIXème siècle.

4.1 Pour aller plus loin

Depuis quelques années, plusieurs groupes se sont formés afin d'ajouter du contenu à propos de divers personnages féminins. C'est le cas par exemple, dans la communauté francophone, de plusieurs ateliers et associations comme "les sans pagEs", "Art+Feminism" ou encore "Let's Fill the Wikipédia Gender Gap" (en français : "Comblons l'écart de genre sur Wikipédia"). En 2017, les articles à propos des femmes étaient de l'ordre de 16% sur le Wikipédia français. Il est difficile de mettre en lien ce chiffre avec nos résultats étant donné que nos données proviennent de plusieurs Wikipédia. On peut néanmoins tenter de l'expliquer de plusieurs façons :

— Le manque de données. Certains individus dans le dataset n'ont pas de genre. Étant donné qu'il est déterminé par la biographie de la personne, il est possible que la proportion de ces personnes avec une biographie trop légère soit majoritairement féminine.

— La place de la femme dans l'histoire : Comme énoncé précédemment, la base contient uniquement des personnes décèdes. Par conséquent, la très grande majorité de nos données proviennent d'époque où la place de la femme n'est pas la même qu'à notre époque.

— Information imprécise : Certains individus dans la base de données ont plusieurs pays dans leur catégorie "Country" et parfois à tort. Cela implique donc qu'ils sont comptés plusieurs fois par notre algorithme puisqu'il n'est pas possible de déterminer quel pays, dans la liste, est associé à sa vraie nationalité (sans oublier la possibilité qu'il soit binational)

5 Difficultés rencontrés

5.1 Récupération des données

La récupération des données est, de loin, le plus gros souci que l'on a dû gérer. Comme énoncé dans le 2.1, on a eu diverses pistes pour récupérer ces données : les `dumps`, le scraping, mais aussi via des query en SPARQL sur le site de Wikidata. Dans tous les cas, on était confrontés à des problèmes. Soit les données étaient beaucoup trop grandes et anarchiques pour être traitées (`dumps`), soit ils en manquaient (scraping, dataset issus de Kaggle), soit on était bloqués dans l'utilisation du service (query wikidata). Étant donné que ce point était le plus important de tous, on a perdus beaucoup de temps à trouver la meilleure solution possible afin de pouvoir rendre le travail le plus juste et intéressant possible.

5.2 Architecture et traitement des données

L'architecture des données a également été une difficulté à surmonter. Comme le graphique que l'on utilise pour afficher nos données ne peut pas lire des informations en profondeur, on a dû réfléchir à des alternatives. La première a été de se servir de la librairie `JSONata`. C'est un paquet permettant de pouvoir réaliser des requêtes sur notre fichier JSON. Ceci permettait donc de pouvoir afficher les informations que l'on souhaite, sans passer par divers calculs ou alors par une architecture peu lisible. Malheureusement, la documentation n'est pas très claire et on n'a pas assez de temps pour approfondir le sujet. On a donc préféré agencer la structure de nos ressources de la manière décrite au 2.2.2. En voici un aperçu :

Et pour afficher la moyenne, comme la fonction `draw` de `bertin` ne peut pas aller en profondeur dans le fichier (ex : l'élément `0` du champs `"100"`), on crée un nouveau champ, nommé `meanGender`. Ce champ est créé (ou modifié) dans la fonction `useEffect`. On récupère l'année que l'utilisateur a choisis et on divise le nombre d'homme (ou de femme) sur le total de personnes de l'année correspondante.

```
realjson.forEach(element => {
  if(element[birthDate][ + reverseViz] === 0)
    element.meanGender = ""
  else
    element.meanGender = (element[birthDate][ + reverseViz]/(element[birthDate][ + reverseViz]+element[birthDate][ + !reverseViz]))*100
})
dataViz = realjson
```

Ici on utilise `reverseViz` pour accéder aux éléments des listes car c'est un booléen et que la longueur des listes et de 2. Ainsi, quand `reverseViz` est à `False` (donc à 0), cela veut dire que l'utilisateur veut afficher la représentation des hommes sur Wikipédia. Donc on fait le nombre d'homme (`element[birthDate][0]`), divisé par le nombre total (homme + femme). Du coup, si jamais `reverseViz` est à `True`, on aura bien le calcul inverse, soit le nombre de femme (`element[birthDate][1]`), divisé également par le nombre total de personnalités.

6 Amélioration possibles

6.1 Qualité des données

L'amélioration majeure que l'on pourrait apporter est bien évidemment le fait de posséder également des ressources sur des personnes existantes. La meilleure solution serait d'améliorer la rapidité de notre script `Crawler.py` puisqu'il n'a aucun filtre. On pourrait aussi bien tomber sur des gens vivants ou morts. Mais il faudrait aussi améliorer sa fiabilité.

Pour améliorer la qualité, il faudrait aussi un dataset un peu plus complet. Bien qu'il soit précis dans la majorité des cas, il y trop d'individus que l'on a écarté par manque d'information (près de 20% soit environ 300 000 éléments).

6.2 Filtres disponibles

Le nombre de filtres que l'on a mis en place est assez faible par rapport aux idées que on a initialement.

Le premier filtre auxquels on a pensé fut un bouton glissant similaire à celui de l'année de naissance des individus mais celui aurait concernés Wikipédia. Il aurait permis, ainsi, de voir l'évolution de la différence des genres depuis la création de Wikipédia, soit de 2003 à nos jours.

Un autre était de pouvoir naviguer en fonction de la langue de Wikipédia. Voir si les inégalités sont les mêmes dans toutes les versions du site ou non, et quel langage est le plus proche de la parité.

On aurait pu également rajouter d'autres professions à étudier (athlète, chercheur, ...) mais leurs quantités n'étaient pas très importantes, et de plus, ce sont des métiers qui ne sont pas intéressant à étudier au cours du temps selon nous.

Enfin, on pourrait offrir une autre vue à l'utilisateur. Une semblable à la partie 4 où l'utilisateur aurait pu naviguer entre différents graphiques et moyennes.

Malheureusement la cause principale de l'absence de ces filtres est le manque de temps et de moyen. Notre script de scraping aurait permis d'avoir accès à ce genre d'information mais, étant trop lent, cela était impossible. En plus de récupérer une base potable de ressources, on aurait dus le faire dans d'autres langue (donc de nouvelle spécificité car le Wikipédia français est différent de Wikipédia anglais, espagnol, ...) et récupérer l'historique des pages visités pour voir leur date de création.

6.3 Liste des personnalités par pays

Une autre amélioration imaginée fut de pouvoir accéder à la liste des personnes analysé dans un pays en cliquant sur ce même pays dans le `svg`. Malencontreusement, il y a deux soucis à la mise en place de ce genre de vue. La première est de pouvoir rendre les pays cliquables. Cela veut donc dire modifier le fichier généré car `bertin` ne propose pas d'option pour réaliser ceci, ce qui est extrêmement compliqué sachant que la balise `svg` est immense est pas prévu pour être lu par un humain. Le second problème est l'architecture des données. Comme énoncé dans le 5.2, `bertin` ne peut pas lire des données en profondeur dans un fichier JSON. Cela aurait nécessité une nouvelle architecture du fichier et encore plus de calcul pour pouvoir produire les chiffres que l'on souhaite.

7 Conclusion

Ce projet fût compliqué à mener de par les diverses difficultés que l'on a rencontré tout du long et l'envie de produire le travail le plus juste et rigoureux possible. Ce projet est améliorable mais permet tout de même de se faire une vision globale de la répartition homme-femme sur Wikipédia. De plus, ce sujet est captivant suite aux multiples action mis en place afin de réduire l'inégalité sur la plateforme. Il serait pertinent de réaliser une nouvelle fois cette expérience dans quelques années, avec une meilleure banque de donnée ou alors un meilleur crawler, pour observer l'évolution (ou non) de la proportion d'article sur des personnages féminin et de l'efficacité des actions visant à réduire l'inégalité homme-femme sur la plateforme.