

# Challenge Dataconnexion - groupe 6.4




## — Projet: FindMeCulture

**Objectif** : Favoriser l'accès à la culture en faisant connaître l'offre culturelle locale à un utilisateur et en le sensibilisant aux inégalités dans ce domaine.

**Lien dépôt Gitlab** : <https://gitlab-cw8.centralesupelec.fr/2019ahvound/projet-coding-weeks.git>

Membres du groupe : David Ah-Voun, Souhir Amri, Marie Desbuquois, Amine Larhchim, Liwa Mansour

# Bases de données utilisées

- 
- Liste et localisation des Musées de France au 31/12/2017 : <https://www.data.gouv.fr/fr/datasets/liste-et-localisation-des-musees-de-france/>
  - Panorama des festivals français : <https://www.data.gouv.fr/fr/datasets/r/4415a028-aa8e-447d-a2e9-d3917b9bd278>
  - Population des régions françaises : <https://www.insee.fr/fr/statistiques/fichier/3677785/ensemble.xls>

# Première fonctionnalité



- Donne des informations sur les musées proches d'une localisation choisie par **l'utilisateur**.
- En entrée : adresse de l'utilisateur, distance maximale à parcourir pour aller au musée
- En sortie : liste de musées situés dans le périmètre choisi, avec les informations nécessaires, y compris si le musée n'a pas de site Internet
- Clientèle ciblée : individus et foyers désirant aller au musée près de chez eux

```

class load_data:
    """
    This class will be used to extract data from databases
    """
    #initialize the class object
    #init flag determines whether this is the first load of the database or not
    #test flag determines whether to get a small sample size or not
    def __init__(self, filename, test=False, init=False):
        self.filename = filename
        if init:
            self.first_load(filename)
            self.load_into_pickle(filename)
        self.data = self.load_from_pickle(filename)
        if test:
            self.test_data = self.data.loc[:10,:]
            self.count_museums = len(self.data)

    def __help__(self):
        print('filename should be without the extension')

    def load_from_pickle(self, filename):
        # loading data from pickle
        data_path = os.path.join(scriptpath, 'data', self.filename+'.pkl')
        data = pd.read_pickle(str(data_path))
        return data

    def load_into_pickle(self, filename):
        # loading data into pickle to save processing time

        data_path = os.path.join(scriptpath, 'data', self.filename+'.xlsx')
        data = pd.read_excel(io = str(data_path))

        pickle_data_path = os.path.join(scriptpath, 'data', self.filename+'.pkl')
        pd.to_pickle(data, str(pickle_data_path))

```

```

def first_load(self, filename):
    # loading data from the excel file directly
    data_path = os.path.join(scriptpath, 'data', self.filename+'.xlsx')
    data = pd.read_excel(io=str(data_path))
    return data

```

```
def find_nearest_museums(address, ville, num_museums=3, max_distance=100000, test=False):

    #load the coordinate dict pickle
    pickle_dir = os.path.join(scriptpath, r"geolocalisation", r"coordinate_dict.pkl")
    with open(pickle_dir, 'rb') as coordinate_pickle:
        coordinates_dict = pickle.load(coordinate_pickle)

    #initilize the adress locator
    fr_locator = BANFrance()

    #get the coordinates and region of the given adress
    address = fr_locator.geocode(address+' '+ville)
    region = address.raw['properties']['context'].split(',')[1].strip()
    address_coordinates = address.raw['geometry']['coordinates']

    # filter the dataframe to avoid processing unlikely candidates
    filtered_df = filter_by_region(df, region)

    #initilize the list of results
    list_of_museums = []

    #initial loop to fill the result list with random entries
    for id, row in filtered_df.iterrows():

        if len(list_of_museums) == num_museums:
            break

        try:

            #get the museum adress, the coordinates
            museum_id = row['ID MUSEE']
            museum_coordinates = coordinates_dict[museum_id]

            #calculate the distance to the given adress
            distance_to_adress = distance(address_coordinates, museum_coordinates).kilomet

        except:
            continue
```

```
        #ignore the museum if it's too far away
        if distance_to_adress >= max_distance:
            continue

        #if all is well, append it to the list of results
        list_of_museums.append((row, distance_to_adress))

    #sort the list of results based on the distance to the adress
    list_of_museums.sort(key=lambda tup:tup[1])

    #if there is not enough data, the next part is unessary so we just return the initial list of results
    if len(list_of_museums) < num_museums:
        return enumerate(list_of_museums)

    #get the distance of the farthest museum
    museum_dist = list_of_museums[-1][1]

    for i, (id, row) in enumerate(filtered_df.iterrows()):

        #the first part of the list of results has already been dealt with
        if i <= num_museums:
            continue

        try:

            #get the museum adress, the coordinates
            museum_id = row['ID MUSEE']
            museum_coordinates = coordinates_dict[museum_id]

            #calculate the distance to the given adress
            distance_to_adress = distance(address_coordinates, museum_coordinates).kilometers


        except:
            continue

        #ignore the museum if it's too far away
        if distance_to_adress > max(museum_dist, max_distance):
            continue

        #remove the unwanted museum
        list_of_museums.append((row, distance_to_adress))
        list_of_museums.sort(key=lambda tup:tup[1])
        list_of_museums.pop()

    return enumerate(list_of_museums)
```

# Deuxième fonctionnalité

- 
- Donne les informations sur des festivals sélectionnés à partir de certains paramètres
  - En entrée : département, mois de l'année et domaine choisis à partir d'une liste
  - En sortie : liste de festivals situés dans le département, se déroulant pendant le mois et associés à ce domaine, des informations complémentaires sont également fournies
  - Clientèle ciblée : individus et groupes désirant organiser à l'avance un voyage dans une zone géographique et pendant une période donnée; personnes qui souhaitent trouver des festivals près de chez eux

# Troisième fonctionnalité



Affiche des diagrammes sur la répartition des musées et des festivals en France :

- Régions, villes où il y a le plus de musées
- Classement des régions en fonction du nombre de musées par habitant, et nombre d'habitants par région
- Régions où il y a le plus de festivals
- Répartition des festivals par domaine
- Répartition des festivals dans l'année ...

Clientèle ciblée : individus intéressés par les inégalités d'accès à la culture; organismes souhaitant lutter contre les déserts culturels (états, associations, organisateurs de festivals, mécènes...)

# Graphes:

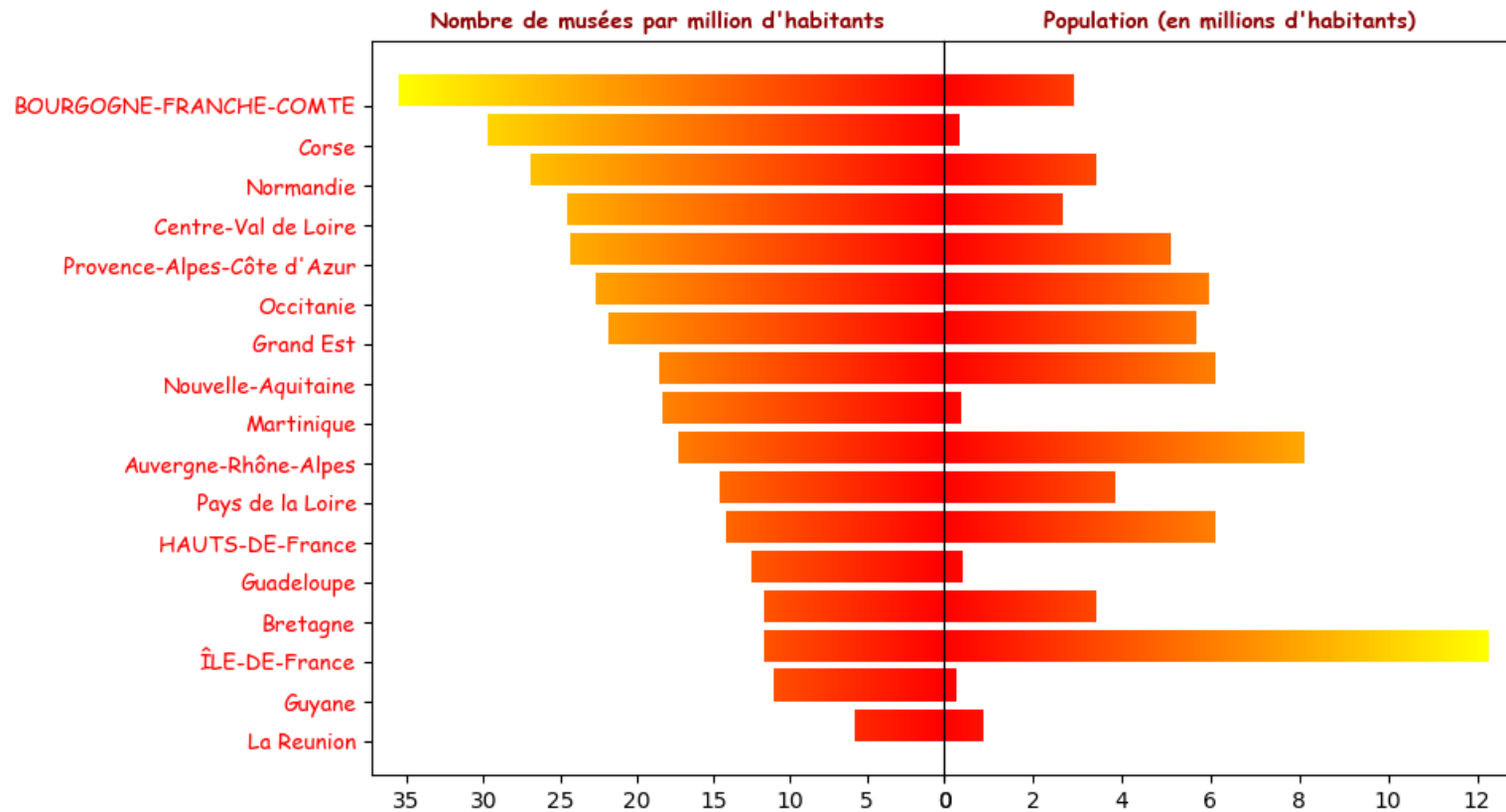
```
def camembert_villes(base_de_donnees,seuil):
    plt.figure(figsize=(12, 6)) #Taille de la figure
    list_of_cities=list_of_cities_seuil(base_de_donnees,seuil)
    taille=[] #Va contenir le nombre de musee de chaque ville dans list_of_cities
    i=0
    for city in list_of_cities:
        nombre_de_musee=len(filtre.filtre_par_villes(base_de_donnees,city).to_numpy()) #On compte le nombre de musée dans la ville
        taille.append(nombre_de_musee)
        list_of_cities[i]= city + '(' + str(nombre_de_musee) + ')' #On écrit le nombre de musee à cote de la ville pour la lisibilité
        i+=1
    list_of_cities.append('moins de '+str(seuil)+' musée') #On rajoute les musées 'autres' qui sont en dessous du seuil
    musee_autre=0 #Va compter le nombre de musée autres
    for city in list_of_cities_seuil(base_de_donnees,0):
        if city not in list_of_cities:
            musee_autre+=len(filtre.filtre_par_villes(base_de_donnees,city).to_numpy())# On somme les musées qui sont dans les villes aut
    taille.append(musee_autre)
    list_of_cities[-1]=list_of_cities[-1]+ '('+ str(musee_autre)+')'
    plt.pie(taille,labels=list_of_cities,textprops={'fontsize': 5})
    plt.title('Répartition des musées selon les villes',fontsize=20,y=0.4)
    plt.show()
```



# Structure du code

## Dossiers et fichiers :

- data : bases de données
  - data extraction : code qui construit les Dataframes avec les fichiers de data
  - GUI : relatif à l'interface graphique
  - Visualization : contient les programmes qui ont tracé les diagrammes
  - Graphes : diagrammes obtenus
  - ancien mvp musée : code du premier mvp, avant la géolocalisation
  - festivals : filtres et programmes de comptage sur la base de données festivals
  - Geolocalisation : programmes liés à la localisation de l'utilisateur et des musées
  - Application Dash : programmes qui construisent le Dash
  - tests : principalement sur les filtres
  - html cov : couverture des tests
- 
- README.md, LICENSE, .gitignore
  - requirements.txt : liste des versions des modules nécessaires
  - main.py : lance l'interface graphique
  - Methodologie.txt : explique les méthodes et raisonnements



# Répartition du travail



Souhir : extraction des données pour les festivals, fonctions de comptage pour les musées et les festivals, une carte des musées, l'application dash

Liwa : Interface graphique, module d'extraction de données, géolocalisation

Amine : Piechart et histogramme montrant la répartition géographiques des musées ( villes, départements,régions), répartition dans l'année des différents festivals

David : Tests avec pytest sur les fonctions de filtrage, histogrammes sur les musées, filtres pour les festivals, histogramme sur les festivals (gradient de couleur, format des textes), (*autre base de donnée traitée mais finalement non pertinente*), rédaction du README et requirements.txt

Marie : filtres géographiques pour les musées, code du MVP initial (l'utilisateur donne une zone et non pas son adresse), tests divers, comptage et diagrammes pour les festivals, rédaction de la méthodologie

 **Merci pour votre attention**

