Manuel Technique de PYstage

Dans l'optique de notre projet consistant à réaliser un outil dynamique d'analyse territoriale et par activité des stages réalisés par les étudiants de l'IUT Poitiers-Niort-Châtellerault et cela à partir d'un fichier Excel qui recense plusieurs données dont les différents stages de l'IUT. Nous avons décidé d'utiliser l'outil Python car nous voulions automatiser le projet le plus possible et cet outil nous donne, via ses librairies, beaucoup de possibilités. Voici le manuel technique expliquant pas à pas les fonctionnalités de notre code, ses résultats et ses limites.

1- Préparation des données

Tout d'abord, Nous importons les librairies Python nécessaires pour notre procédure :

- « Panda » pour les Data Frame ;
- « Json » pour lire les fichiers Json ;
- « webbrowser » afin d'ouvrir une page Html ;
- « os » pour vérifier l'existence d'un fichier ;
- « Numpy » pour les tableaux dans la console ;
- « Tkinter » pour l'interface ;
- « Customtkinter » pour styliser l'interface ;
- « Folium » pour la création de cartes ;
- « plotly.express » pour la création autres graphiques ;
- « PIL Image », « PIL ImageDraw » et « PIL ImageFont » pour la création d'image sur HTML.

Il faudra installer dans la console (en bas à gauche) les librairies customtkinter, plotly et folium afin de les utiliser, et il se peut que xlrd et openpyxl ne soit pas installés si vous utilisez une version antérieure de Spider/Python donc il faudra l'installer par la même procédure.

Ensuite nous voulions importer le fichier Excel dans Python, et nous avons remarqué que les fichiers sont beaucoup plus faciles à utiliser en format .csv, donc nous avons converti le fichier et dans le cas où le fichier est déjà dans le format attendu, le fichier sera simplement lu.

```
# Permet de créer un fichier csv à  partir d'un excel
datacsv = 'Base_AREXIS.csv'

#Si le fichier excel existe déjà alors on le lis
if os.path.isfile(datacsv):
    arexis = pd.read_csv('Base_AREXIS.csv', sep=';')

#Sinon on créer un fichier csv à partir du fichier excel. Cela permet d'améliorer la fluidité de notre outil
else:
    arexis = pd.read_excel('Base_AREXIS.xlsx')
    arexis.to_csv ("Base_AREXIS.csv", sep=";")
```

En observant le fichier Excel, nous avons remarqué que l'intitulé des formations était beaucoup trop divers pour être exploité, nous avons donc créé des dictionnaires avec des formations plus adéquats afin de les regrouper et les exploiter.

Ensuite, on a créé une fonction « variabiliser » qui sert à regrouper toutes les formations annexes dans les formations générales que l'on a au préalable renseigné dans le dictionnaire. La fonction prend en entrée 4 informations :

1- Le data Frame (données stockées en deux dimensions, comme un tableau) dans laquelle se trouve les données à regrouper.
2- Le nom de la colonne qu'il faut regrouper
3- Le dictionnaire avec les nouveaux entêtes
4- Le nom de la nouvelle colonne créé par la fonction

La fonction ressort ensuite un nouveau data frame avec les données regroupées dans les nouveaux entêtes, évidemment la fonction renvoie une erreur si vous sélectionnez une simple liste/dictionnaire à la place d'un data frame, si le nom de la colonne n'est pas dans le data frame ou si le dico de la variable ne contient rien.

```python
#permet de regrouper des variables selon un dictionnaire établi
def variabiliser(df,nomDeColonne,dicoVariables,nomNouvelleColonne):
    variablesADegager = df[nomDeColonne]
    listeNouvelleVariable=[0]*len(variablesADegager)
    index=-1
    for uneVariable in variablesADegager:
        index+=1
        for laVariable in dicoVariables:
            if isinstance(dicoVariables[laVariable], list):
                for unElement in dicoVariables[laVariable]:
                    if unElement in uneVariable:
                        listeNouvelleVariable[index]=laVariable
                    else:
                        if listeNouvelleVariable[index]==0:
                            listeNouvelleVariable[index]='Autres'
            else:
                if dicoVariables[laVariable] in uneVariable:
                    listeNouvelleVariable[index]=laVariable
                else:
                    if listeNouvelleVariable[index]==0:
                        listeNouvelleVariable[index]='Autres'
    dfFinal = df
    dfFinal[nomNouvelleColonne]=listeNouvelleVariable
    return dfFinal
```

On enlève les valeurs non renseigné (écrites « NaN » dans Python), on remplace les entêtes qui nous sembles nécessaires de remplacer et on utilise la fonction du dessus pour regrouper les formations.

```
#Enlever les nan qu'on a remplacer par du vide
arexis.fillna('',inplace=True)

arexis["type_diplome"] == arexis['type_diplome'].map({"DUT01":"DUT1"})

#Remplacement des DUT01 et DUT02 par DUT1 et DUT2
replaceDUT01 = {'DUT01':'DUT1'}
arexis = arexis.replace(replaceDUT01)

replaceDUT02 = {'DUT02':'DUT2'}
arexis = arexis.replace(replaceDUT02)

# Le data frame est transposé dans un tableau numpy
tableauData = np.array(arexis)

#Permet d'affecter la fonction sur notre data Frame initial afin d'avoir les formations regrouper
data2 = variabiliser(arexis,'intitule',dicoFormation,'formationGen')
```

On créer une liste dans laquelle on enlève les doublons des années et on rajoute la valeur « All ».

```
#On regarde pour l'année toute les valeurs unique afin de les rassembler
annee_double = np.array(arexis["idexercice"])
annee_double2 = pd.unique(annee_double)


#On crée une liste dans laquelle on va ajouter chaque année unique ainsi qu'une selection all
OptionListAnnee =  ["ALL"]
app = tk.Tk()

#On ajoute chaque année unqiue à  notre liste contant le all
for i in range(len(annee_double2)):

    OptionListAnnee.append(annee_double2[i])
```

2- Création de l'interface

Afin d'optimiser au mieux notre outil, il nous fallait un moyen simple mais efficace de savoir avec certitude les informations recherchées par l'utilisateur, nous avons donc opté pour une interface Tkinter avec 3 filtres qu'il faut sélectionner.

On commence par définir la page Tkinter, les dimensions, la possibilité de l'agrandir ou non. Ici on a choisi une dimension bien précise car elle est la même que celle de l'image de font car si l'on grandi ou rétréci la page, tous les widgets présents sur le tkinter se décalent, et le bouton qui sert à quitter l'interface en particulier apparait 2 fois sur l'écran.

Pour une interface plus soignée on décide de placer les barres de défilement avant l'image car nous n'avons pas réussi à modifier son aspect, ainsi celle-ci sera utilisable malgré tout grâce au scrolling de la souris mais non visible.

Enfin on importe l'image de fond que l'on a préparé aux dimensions de la page tkinter. À noter que nous avons choisi la fonction « PhotoImage » est l'une des plus pratique car utilisable sur plein de formats connus (png, jpeg, gif, tif, ppm, bmp etc…)

```
#Formatage de la page

app = tk.Tk()
app.title('Interface PYstage')
app.geometry('800x700')
app.resizable(width=0, height=0)
app.configure(bg='#C5D1DC')
app.iconphoto(False, tk.PhotoImage(file='PYstage.png'))

yscrollbarAnnee = Scrollbar(app, repeatdelay = 0, bg = "#E4C79B", bd= 0,highlightthickness = 0,highlightcc
yscrollbarAnnee.pack()
yscrollbarAnnee.place(x=230,y=480,width=13, height=100)

yscrollbarformation = tk.Scrollbar(app, repeatdelay = 0)
yscrollbarformation.pack()
yscrollbarformation.place(x=480,y=480,width=13, height=100)
yscrollbarformation.config (background ='#1F6AA4', borderwidth = 0, relief  = 'flat')

yscrollbarDiplome = tk.Scrollbar(app, background ='#1F6AA4', borderwidth = 0, repeatdelay = 0)
yscrollbarDiplome.pack()
yscrollbarDiplome.place(x=755,y=480,width=13, height=100)
```

On construit ensuite les listes de sélections des filtres, avec la fonction « Listbox » dans laquelle on doit définir la page tkinter dans laquelle elle doit s'afficher, son mode de sélection (si dessous « multiple » est précisé car on doit pouvoir prendre plusieurs année), sa taille, la barre de sélection qui sera utilisé dans la liste. On peut aussi choisir dans cette fonction plusieurs éléments de style comme la couleur, le relief ou encore la police).

On insert ensuite les années que l'on a précédemment récupéré dans un dictionnaire et on place ses coordonnées dans le tkinter (x pour la longueur depuis le coin gauche et y la hauteur depuis le coin supérieur) et pour finir on lie la barre de défilement à la liste de sélection.

```
optAnnee = Listbox(app,selectmode = "multiple",height=7, yscrollcommand = yscrollbarAnnee.set,exportselection = False, actives
for f in range(len(OptionListAnnee)):
    optAnnee.insert(f,OptionListAnnee[f])
    optAnnee.pack()
    optAnnee.place(x = 50,y = 280)
    compteur += 1
yscrollbarAnnee.config(command = optAnnee.yview)
#Création d'une liste dans lequel on considére que chaque élèment sont cliqués, cette liste sera affectées à  l'option all
allitemAnnee = []
allitemAnnee += optAnnee.get(1,compteur)


lstAnnee = []
```

Pour chaque liste de sélection on y associe une fonction qui récupère les choix réalisés par l'utilisateur et qui gère la sélection de la liste grâce à « itemconfig » (si l'utilisateur choisi « tous » puis une autre formation par exemple, la sélection de « tous » ne sera pas retenu et vice-versa).

```
#Création d'une fonction qui retranscrit les élèments sélectionnez de notre listobx
def selected_Annee():
    for i in optAnnee.curselection():
        if optAnnee.get('active') == 'All':
            lstAnnee.insert(i,allitemAnnee)
            optAnnee.selection_clear(1,'end')
            optAnnee.itemconfig(i,foreground = 'lightgray')
            optAnnee.itemconfig(0, foreground = "Black")
        else:
            print(optAnnee.get(i))
            lstAnnee.insert(i, optAnnee.get(i))
            optAnnee.selection_clear(0)
            optAnnee.itemconfig(0,foreground = 'lightgray')
            optAnnee.itemconfig(i, foreground = "Black")
```

Ensuite on place les boutons avec la même fonction dans lequel on renseigne la page tkinter, le titre du bouton, sa fonctionnalité (ici le bouton ferme la page tkinter) ainsi que d'autres informations comme la police, la couleur ou le relief.

```
quitte = tk.Button(app, text="Quit", command=app.destroy, font = ("Helvetica",'12',"bold"))
quitte.pack()
quitte.place(x=900,y=10)
```

Grâce au compteur initié au préalable on peut récupérer dans le Data Frame les informations en fonction des filtres que l'utilisateur a sélectionnées.

```
lstFormation = []
allitemFormation = []
allitemFormation += optformation.get(1,compteur)


diplome = np.array(arexis["type_diplome"])
diplome_double = pd.unique(diplome)

OptionListDiplome  =  ["Tous"]
```

On récupère tous les filtres quand l'utilisateur choisit « tous » dans l'interface tkinter et l'on crée un data frame avec les filtres appliqués (on récupère les données grâce au Data Frame créé précédemment).

```
for i in range(len(lstAnnee)):
    lstAnnee += lstAnnee[i]

for i in range(len(lstDiplome)):
    lstDiplome += lstDiplome[i]

for i in range(len(lstFormation)):
    lstFormation += lstFormation[i]


# Création des data frame avec les filtres appliqués

filtre = arexis.query("idexercice == " + str(lstAnnee) + '')

filtre2 = arexis.query("formationGen == " + str(lstFormation) + '' )

filtre3 = arexis.query("type_diplome == " + str(lstDiplome) + '')

final = pd.merge(filtre, filtre2)
final2 = pd.merge(final, filtre3)
```

Ensuite on transforme le code NAF renseigné dans le fichier Excel en secteur auquel ils correspondent (les secteurs 4 6 et 8 correspondent à des secteurs principaux donc on prend les 3 premiers caractères, sinon on en prend qu'un).

```
#gestion de la transformation du code NAF en secteur
df=final2
secteurNAF = df['naf']
listeSecteur = []
for unSecteur in secteurNAF:
    if  unSecteur!='':
        if str(unSecteur)[0]=='4' or str(unSecteur)[0]=='6' or str(unSecteur)[0]=='8':
            listeSecteur.append(str(unSecteur)[0:2])
        else:
            listeSecteur.append(str(unSecteur)[0])
    else:
        listeSecteur.append('n')
```

On créer ensuite la fonction « creerDataFrame » qui, à partir d'un data Frame et du nom de la colonne nous donne un data Frame composé des colonnes et du nombre de stages.

```
#permet de compter le nombre de stages en fonction d'un paramètre
def creerDataFrame(df,nomdeColonne):
    df2 = df.drop_duplicates(subset = [nomdeColonne])
    colonne = df[nomdeColonne]
    colonne2 = df2[nomdeColonne]
    nbColonne = [0] * len(colonne2)
    listeVariables = []
    index = -1
    for laFormation in colonne2:
        index+=1
        listeVariables.append(laFormation)
        for uneFormation in colonne:
            if laFormation == uneFormation:
                nbColonne[index]+=1
    df3 = pd.DataFrame(
        {'Colonne':listeVariables,
         'Nombre de stages': nbColonne
        })
    return df3
```

On crée dans le même temps les fonctions « enleverValeur » et « garderValeur » qui enlève ou conserve une valeur précise d'un data Frame.

```
#sert d'enlever à un dataframe une valeur précise
def enleverValeur(df,colonne,valeur):
    indexNames = df[df[colonne]==valeur].index
    df2 = df.drop(indexNames)
    return df2
```

```
#sert de garder une valeur précise d'un dataframe
def garderValeur(df,colonne,valeur):
    indexNames = df[df[colonne]!=valeur].index
    df2 = df.drop(indexNames)
    return df2
```

On récupère ensuite la colonne secteur du data frame, on crée ensuite un dictionnaire des secteurs en fonctions des codes NAF correspondants. On utilise ensuite les fonctions « variabiliser » « créerDataFrame » et « enleverValeur » pour créer un data Frame qui ajoutera une colonne secteur au data frame entré en paramètre.

La même procédure est faite pour le code du département

```
df['codeSecteur']=listeSecteur
dicoSecteur = {"Emploi":["86","7"],"Industriel":["1","2","3"],"BTP":["41","42","43"],"Commerce":["45","46
dfFiltré = variabiliser(df,'codeSecteur',dicoSecteur,'secteur')
dfFin = creerDataFrame(dfFiltré,'secteur')
dfFinFin1 = enleverValeur(dfFin, 'Colonne', 'Erreur')

codeNAF = dfFinFin1['Colonne']
nomEntreprise=[]
nbStructure=[]
total=0
for unCode in codeNAF:
    data = garderValeur(dfFiltré,'secteur',unCode)
    nomEntreprise.append(Top(data,'denom_social','Exercice',1)['nom_entreprise'][0])
    nbStructure.append(len(creerDataFrame(data,'denom_social')['Nombre de stages']))
dfFinFin1['topEntreprise']=nomEntreprise
dfFinFin1['nbStructure']=nbStructure
```

3- Création des graphiques

Pour la création des graphiques nous avons décidé d'utiliser les librairies plotly et folium (pour les cartes). Nous avons choisi ces deux-là car elles nous permettent de garder l'interactivité et peuvent se visualiser dans une page html.

En premier temps avec plotly on a créé 2 graphiques circulaires, 1 pour les codes NAF et le second pour les départements. On a également fait un diagramme en barre qui montre la répartition des stages en fonction des formations. Pour pouvoir afficher les différentes informations tels que le nombre de

stages ou encore la top structures sur chacun des graphiques nous avons eu beaucoup recourt aux différentes fonctions. C'est également à cet endroit qu'on peut modifier la taille des graphiques.

Nous avons également, pour chacun des graphiques, géré les différentes erreurs possibles, notamment lors de filtres non compatibles ce qui nous afficheras des graphiques vides sans données.

Les différentes fonctions qu'on a utilisé pour les graphiques sont dessinerBar, dessinerCamembert et dessinerLigne. Ces fonctions nous permettent de créer un graphique à partir du dataframe que l'on désire et sur les colonnes que l'on veut. C'est dans la fonction qu'on définit les titres des axes, les couleurs ainsi que différents éléments graphiques.

```
def dessinerBar(df,descending,repartition,annee):
    if type(annee) is list:
            titre = "Répartiton des " + str(repartition) + " sur toutes les années"
    else:
        titre = "Répartiton des " + str(repartition) + " en " + str(annee)
    couleur = ['#d39c83']*len(df)
    fig = px.bar(df, x='Colonne',
                 y='Nombre de stages',
                 color_discrete_sequence=couleur,
                 hover_data=['topEntreprise','nbStructure']
                 )
    fig['layout']['xaxis']['autorange'] = "reversed"
    fig.update_layout(title={
                            'text':titre,
                            'y':0.97,
                            'x':0.5,
                            'xanchor': 'center',
                            'yanchor': 'top'})
    if descending:
        fig.update_layout(barmode='stack', xaxis={'categoryorder':'total descending'})
    fig.write_html("Html\\barchart.html")

#permet de dessiner un diagramme en ligne
def dessinerLigne(df):
    fig = px.line(dfFin, x="Année", y="Nombre de stages",width=750,height=700,color='Formation',markers= True,title="Évolution du nombre de st
    fig['layout']['xaxis']['autorange'] = "reversed"
    fig.update_layout(title_x=0.5,paper_bgcolor="#817D7D")
    fig.update_traces(line=dict(width=4),marker=dict(size=12))
    fig.write_html("Html\\ligne.html")

#permet de dessiner un pie chart
def dessinerCamembert(df,plus,dimension,repartition,annee):
    if type(annee) is list:
            titre = "Répartiton des " + str(repartition) + " sur toutes les années"
    else:
        titre = "Répartiton des " + str(repartition) + " en " + str(annee)
    fig = px.pie(df, values='Nombre de stages', names='Colonne',
             title=titre,
             custom_data=['topEntreprise','nbStructure'], labels={'topEntreprise':'Meilleure structure','nbStructure':'Nombre de structures'},
             color_discrete_sequence=px.colors.sequential.Brwnyl,
             width=dimension[0],
             height=dimension[1])
    fig.update_traces(textposition='inside', textinfo='percent+label',\
                hovertemplate = "Departement:%{label}: <br>Nombre de stages: %{value} </br>(Meilleure structure, Nombre de structures) : %{cus
    fig.update_layout(title_x=0.5)
    nom_html = "html\\camembert"+str(plus)+".html"
    fig.write_html(nom_html)
```

```
#gestion des erreurs en cas de dataframe vide
if len(final2)!=0:
    if lstAnnee != []:
        dessinerCamembert(dfFinFin1,'',[630,380],'structures',lstAnnee[0])
        dessinerCamembert(dfFinFin2,1,[630,380],'departements',lstAnnee[0])
        dessinerBar(dfFinFin,True,'formations',lstAnnee[0])
else:
    ar = np.array([[0,0,0,0], [0,0,0,0], [0,0,0,0]])
    prout = pd.DataFrame(ar, columns = ['Colonne','Nombre de stages','topEntreprise','nbStructure'])
    dessinerCamembert(dfFinFin1,'',[630,380],'structures','')
    dessinerCamembert(dfFinFin2,1,[630,380],'departements','')
    dessinerBar(prout,True,'formations','')
```

Ensuite nous avons créé le graphique linéaire qui montre le nombre de stages par formation sur toutes les années. Pour cela, on a de nouveau utilisé la fonction dessinerLigne. On a donc fait un

dataFrame avec uniquement les colonnes qui nous intéressait, donc les années, les formations et une colonne qui compte le nombre de stages par formation.

```
#creation du dataframe du nombre de stages par formation en fonction de l'année pour diagramme en ligne
df = variabiliser(arexis,'intitule',dicoFormation,'formationGen')
dfFiltré = creerDataFrame(df,'formationGen')
formation = dfFiltré['Colonne']
perkastor=0
for uneFormation in formation:
    data = garderValeur(df,'formationGen',uneFormation)
    if perkastor == 0:
        perkastor=1
        dfFin = creerDataFrame(data, 'idexercice')
        dfFin=dfFin.assign(Formation=uneFormation)
        dfFin.columns = ['Année', 'Nombre de stages','Formation']
    else:
        dfInter = creerDataFrame(data, 'idexercice')
        dfInter=dfInter.assign(Formation=uneFormation)
        dfInter.columns = ['Année', 'Nombre de stages','Formation']
        dfFin = pd.concat([dfFin, dfInter],ignore_index = True,sort = False)

dessinerLigne(dfFin)
```

Pour la partie création de cartes avec la librairie folium, nous avons de nouveau fait appel à des fonctions. Chaque fonction est liée à chaque carte (pays, région, commune). Tout comme pour les graphiques c'est dans ces dernières que l'on formate les éléments graphiques comme la couleur des échelles ou le titre des axes.

```
def add_data_GeoJson_Monde(map):
    commune_data = folium.features.GeoJson(
        data=pays,
        style_function=style_function,
        highlight_function=highlight_function,
        tooltip=folium.features.GeoJsonTooltip(
            fields=["name","nbstage","topstructure","nbstructure"],
            aliases=["Pays : ","Nombre de stage : ","Structure avec le plus d'embauche : ","Nombre de structur
            style=(
                "background-color: white; color: #333333; font-family: arial; font-size: 12px; padding: 10px;"
            ),
            sticky=False,
        ),
    )
    map.add_child(commune_data)
    map.keep_in_front(commune_data)

def add_data_GeoJson_Region(map):
    commune_data = folium.features.GeoJson(
        data=region,
        style_function=style_function,
        highlight_function=highlight_function,
        tooltip=folium.features.GeoJsonTooltip(
            fields=["nom","nbstage","topstructure","nbstructure"],
            aliases=["Région : ","Nombre de stage : ","Structure avec le plus d'embauche : ","Nombre de struct
            style=(
                "background-color: white; color: #333333; font-family: arial; font-size: 12px; padding: 10px;"
            ),
            sticky=False,
        ),
    )
    map.add_child(commune_data)
    map.keep_in_front(commune_data)
```

```python
def add_data_GeoJson(map):
    commune_data = folium.features.GeoJson(
        data=commune,
        style_function=style_function,
        highlight_function=highlight_function,
        tooltip=folium.features.GeoJsonTooltip(
            fields=["nom","codepostal","nbstage","topstructure","nbstructure"],
            aliases=["Commune : ","Code postal : ","Nombre de stage : ","Structure avec le plus d'embauche : "
            style=(
                "background-color: white; color: #333333; font-family: arial; font-size: 12px; padding: 10px;"
            ),
            sticky=False,
        ),
    )
    map.add_child(commune_data)
    map.keep_in_front(commune_data)

add_data_GeoJson(communemap)

folium.LayerControl().add_to(communemap)
```

Pour commencer, chaque carte doit avoir un fichier json contenant les coordonnées géographiques des contours des pays, région ou communes, on l'ouvrira sous forme d'un dictionnaire. Mais aussi un code/id pour pouvoir faire le lien entre le fichier json et le dataframe expliqué plus tard. Pour faire une carte on doit aussi avoir un dataframe à deux colonnes contenant dans le même ordre que le json tous les codes/ids, et une valeur dans notre cas ça sera le nombre de stage.

Sur cette première capture on récupère les ids des pays du fichier json (listePays). Puis aussi ceux présents dans le dataframe (df) qui compte le nombre de stage par pays. Les noms des pays dans l'array (nomPays), et pour le nombre de stage dans l'array (nbstage). On transforme ces deux array en liste, nbstage en stageAssocPresents et nompays en paysPresents.

Comme on l'a dit précédemment il faut mettre le nombre de stages de chaque pays dans le même ordre que les pays dans le fichier json. Pour cela on va parcourir la liste des pays du fichier json et les pays présents dans le dataframe. On va ensuite les comparer, dès que les deux sont pareils on va ajouter dans une liste au même index que les pays présent dans le fichier json le nombre de stage. On a ensuite le nombre de stage par pays dans le même ordre que le fichier json on créer ensuite un dataframe (dffifi).

```
#########################              nb stage              ################################

pays = json.load(open("./carte/monde.geojson"))

df = creerDataFrame(final2,"idpays_service")

listePays = [i["id"] for i in pays["features"]]
stagesAssoc = [0] * (len(listePays))

nbstage = df["Nombre de stages"]
nompays = df["Colonne"]
stagesAssocPresents = []
paysPresents = []

for unStage in nbstage :
    stagesAssocPresents.append(unStage)

for unPays in nompays :
    paysPresents.append(unPays)

compteur = 0
compteur2 = 0

for aaa in listePays :
    for bbb in paysPresents :
        if str(aaa) == str(bbb) :
            stagesAssoc[compteur] = stagesAssocPresents[compteur2]
        compteur2 += 1
    compteur +=1
    compteur2 = 0

dffifi = pd.DataFrame({
    "Pays" : listePays,
    "Unite" : stagesAssoc
    })
```

Après cela on arrive à avoir une carte qui est délimitée avec les coordonnées GPS et qui représente le nombre de stages, mais pas d'interactivité. Il faut afficher au passage de la souris, le nom du pays, le nombre de stages, la structure avec le plus de stages, et le nombre de structures d'accueil.

Pour cela il faut ajouter dans le dictionnaire (qui contient notre fichier json), les 4 données. Il faut les ajouter dans le properties de chaque pays sous la forme d'un dictionnaire (clé : valeur), exemple : (nbstructure : 5).

On a déjà le nom du pays et le nombre de stage il suffit juste de mettre les deux infos dans un dictionnaire et ajouter ces deux dictionnaires dans le properties de chaque pays. Grâce à ces deux boucles.

```
pret2 = []
for az in range (0,len(listePays)) :
    dic2 = { "topstructure" : listeStruc[az]}
    pret2.append(dic2)

for aa2 in range(len(pret2)) :
    pays["features"][aa2]["properties"].update(pret2[aa2])
```

```
########################                top structure           ##########################

listeStruc = []
for un in listePays :
    if un in paysPresents :
        dataframe = garderValeur(final2, 'idpays_service', un)
        top = Top(dataframe,'denom_social','Exercice',1)
        toptop = top["nom_entreprise"]
        for unun in toptop :
            listeStruc.append(unun)
    else :
        listeStruc.append("Pas de donnée")

pret2 = []
for az in range (0,len(listePays)) :
    dic2 = { "topstructure" : listeStruc[az]}
    pret2.append(dic2)

for aa2 in range(len(pret2)) :
    pays["features"][aa2]["properties"].update(pret2[aa2])


########################          nombre de structure d'acceuil        ####################
nbStruc = []
for prout in listePays :
    if prout in paysPresents :
        dataframe = garderValeur(final2, 'idpays_service', prout)
        liste = dataframe['denom_social']
        listev2 = []
        for undenom in liste :
            listev2.append(undenom)
        listev3 = np.unique(listev2)
        sheeesh = len(listev3)
        nbStruc.append(sheeesh)
    else :
        nbStruc.append("0")

pret3 = []
for azz in range (0,len(listePays)) :
    dic3 = { "nbstructure" : nbStruc[azz]}
    pret3.append(dic3)

for aa3 in range(len(pret3)) :
    pays["features"][aa3]["properties"].update(pret3[aa3])
```

Pour le top structure on utilise la fonction top qui revoie la top structure. C'est mis sous forme d'une liste puis ajouté dans properties come avant. Pour le nombre de structure d'accueil on va juste récupérer toutes les structures par pays puis enlever les doubles et compter, puis les mettre dans une liste et on les met en dictionnaire pour les ajouter après dans properties.

Il ne nous reste donc plus qu'à formater la carte, c'est-à-dire mettre les différentes informations vues précédemment sur l'affichage et les différentes couleurs. C'est ici que l'on peut changer le zoom-start c'est-à-dire le premier aperçu que l'on a sur la carte (soit carte du monde, de la France ou des communes). Nous avons également légendé notre échelle afin qu'elle soit compréhensible.

Geo_data est le dictionnaire contenant le fichier json

Data = dataframe avec id pays et nb stage

Columns est le nom des colonnes du dataframe(dffifi)

Key_on est le champ commun entre geo_data et data

```
####################       carte       ####################

monde = folium.Map(location = [30, 15],zoom_start=1)

folium.Choropleth(
    geo_data = pays,
    data = dffifi,
    name="choropleth",
    columns = ["Pays","Unite"],
    key_on="feature.id",
    color = "blue",
    fill_color='YlOrRd',
    legend_name='Nombre de stages',
    fill_opacity=0.6,
    line_opacity=0.5,
).add_to(monde)

# folium.LayerControl().add_to(monde)



style_function = lambda x: {
        "color": "#FFFFFF",
        "fillOpacity": 0,
        "weight": 0.1,
    }

highlight_function = lambda x: {
    "fillColor": "#000000", #couleuru tout
    "color": "#000000",   #couleur des bords
    "fillOpacity": 0.75,
    "weight": 0.3,
}




add_data_GeoJson_Monde(monde)

folium.LayerControl().add_to(monde)
```

```
monde.save("html\\onycroit.html")
```

Après cela nous mettons simplement la carte dans une page html qui sera appelée à partir d'une balise iframe dans notre page html finale.

Toutes les procédures qui ont été faites pour la carte monde ont également été faites pour la carte des régions ainsi que la carte des communes. Donc on va juste expliquer ce qui est différent

Pour les régions il fallait regrouper les départements par région car dans le fichier Excel aucune colonne nous permet d'avoir les régions. On créer des listes contenant les 2 premiers chiffres de chaque département pour chaque région. Ensuite on ajoute à notre dataframe contenant notre base de donnée une autre colonne avec le numéro de région

```
region = json.load(open("./carte/regions.geojson"))


data2 = garderValeur(final2, "nom_pays_court", 'FRANCE')

liste11 = ['75','77','78','91','92','93','94','95']
liste24 = ['18','28','36','37','41','45']
liste27 = ['21','25','39','58','70','71','89','90']
liste28 = ['14','27','50','61','76']
liste32 = ['59','60','62','80']
liste44 = ['08','10','51','52','54','55','57','67','68','88']
liste52 = ['44','49','53','72','85']
liste53 = ['22','29','35','56']
liste75 = ['16','17','19','23','24','33','40','47','64','79','86','87']
liste76 = ['09','11','12','30','31','32','34','46','48','65','66','81','82']
liste84 = ['01','03','07','15','26','38','42','43','63','69','73','74']
liste93 = ['04','05','06','13','83','84']
liste94 = ['02']


a = data2["cp_service"]
a = list(a)
listeDepartements = []
for un in a :
    deux = un[:2]
    listeDepartements.append(deux)
```

```
oups = []
for un in listeDepartements :
    if un in liste11 :
        oups.append('11')
    elif un in liste24 :
        oups.append('24')
    elif un in liste27 :
        oups.append('27')
    elif un in liste28 :
        oups.append('28')
    elif un in liste32 :
        oups.append('32')
    elif un in liste44 :
        oups.append('44')
    elif un in liste52 :
        oups.append('52')
    elif un in liste53 :
        oups.append('53')
    elif un in liste75 :
        oups.append('75')
    elif un in liste76 :
        oups.append('76')
    elif un in liste84 :
        oups.append('84')
    elif un in liste93 :
        oups.append('93')
    elif un in liste94 :
        oups.append('94')
    else :
        oups.append('')

data2['cp2'] = oups
```

Ensuite le reste reste pareil que la carte du monde on fait exactement la même chose.

Pour les communes c'est différent car le fichier Excel contient les codes postaux et le fichier json (commune) les codes Insee, il a donc fallu ajouter a chaque code postal du Excel un code Insee pour cela, on a trouver un autre fichier json (merge) faisant le lien entre les deux. Pour cela on commence par enlever toutes les communes qui ne nous intéresse pas c'est-à-dire celles dont les deux premiers chiffres ne commencent pas par la listeDep. Ensuite on récupère la liste des code postaux du fichier json et du Excel dans deux listes différentes. Après on les parcoure et on les compare s'ils sont égaux on ajoute dans leur properties le code Insee correspondant.

```
commune = json.load(open("./carte/communes.geojson"))
merge = json.load(open("./carte/communecp.json"))
data = pd.read_csv('Base_AREXIS.csv', sep=';')
listeDep = [16,17,86,79,33,24,19,23,24,40,47,64,87]
merge2 = []

for uneCommune in merge:
    if round(uneCommune['Code_postal']/1000,0) in listeDep:
        merge2.append(uneCommune)

merge=merge2
listeCodePostal = []
b = []
for i in merge :
    listeCodePostal.append(i['Code_postal'])
    b.append(i['Code_commune_INSEE'])

acom = []
for i in range(len(commune["features"])):
    acom.append((commune["features"][i]["properties"]["code"]))

aacom = [0] * len(acom)


compteur = 0
for i in acom :
    for ii in b :
        if str(i) == str(ii) :
            ish = b.index(ii)
            aacom[compteur] = str(listeCodePostal[ish])
    compteur += 1
```

Ensuite on fait la même chose que pour la carte du monde et des régions.

Maintenant que nous en avons fini avec les cartes on s'est penché sur les différentes informations que vous pouvez apercevoir sur la page home de notre outil.

Dans un premier temps on s'est occupé des médailles. On a donc importé une image avec 3 médailles (or, argent, bronze) et nous avons défini un positionnement pour que les noms des entreprises s'affichent. Cet indicateur est également lié au filtre et peut donc ne contenir aucune donnée si les filtres ne sont pas compatibles. Dans ce cas nous avons décidé d'inscrire « données manquantes » à la place du nom des entreprises.

```
#Creation des images pour les medailles

if len(final2)!=0:
    medailles = Top(final2,"denom_social","Exercice",3)
else:
    ar = np.array([["DONNEES MANQUANTES", 0], ["DONNEES MANQUANTES", 0], ["DONNEES MANQUANTES", 0]])
    medailles = pd.DataFrame(ar, index = ['0', '1', '2'], columns = ['nom_entreprise','Exercice'])

img = Image.open('./image/Medailles.jpg')

I1 = ImageDraw.Draw(img)

Place1 = medailles["nom_entreprise"][0]
Place2 = medailles["nom_entreprise"][1]
Place3 = medailles["nom_entreprise"][2]

myFont = ImageFont.truetype("Roboto-BlackItalic.ttf", 30)

I1.text((200, 100), Place1, fill=(0, 0, 0), font=myFont)
I1.text((200, 280), Place2, fill=(0, 0, 0), font=myFont)
I1.text((200, 460), Place3, fill=(0, 0, 0), font=myFont)

img.save("./image/car2.png")
```

Nous avons ensuite réalisé notre deuxième indicateur. Celui-ci consiste à afficher des drapeaux sur un podium en fonction des 3 pays qui ont reçu le plus de stagiaires (hors France). Bien entendu cet indicateur change en fonction de vos choix de filtres. Pour cela on a dans un premier créé un dataFrame qui enlève la France. On a ensuite regardé quels étaient les 12 pays les plus concernés par les stages et nous avons importé une image de drapeau pour chacun de ces pays. Par la suite nous avons importé une image d'un podium et défini des positions pour chacun des 3 drapeaux sur le podium. On regarde ensuite quels sont les 3 pays ayant accueilli le plus de stagiaires en fonction des filtres. Comme nous n'avons pas importé chaque image pour chaque pays il se peut que certains drapeaux n'apparaissent pas sur le podium. Ils seront remplacés par un drapeau blanc. Également si vos filtres ne sont pas compatibles, des drapeaux blancs s'afficheront.

```
drapeau = enleverValeur(final2,"nom_pays_court","FRANCE")



best = Top(drapeau,'nom_pays_court','Exercice',3)



dfo = best

dfo.drop(dfo[dfo['nom_entreprise'] == ''].index , inplace=True)


rajout_de_lignes = pd.DataFrame([[""]], columns=['nom_entreprise'])
dfo = pd.concat([dfo,rajout_de_lignes], ignore_index=True)
dfo = pd.concat([dfo,rajout_de_lignes], ignore_index=True)
dfo = pd.concat([dfo,rajout_de_lignes], ignore_index=True)

listePaysPodium = ["CANADA", "ETATS_UNIS","ROYAUME_UNIS","ITALIE","ESPAGNE","NORVEGE","PORTUGAL","CHINE","ALLEMAGNE","BELGIQ

CANADA=Image.open('./image/canada.png')
ETATS_UNIS=Image.open('./image/ETATS_UNIS.png')
ROYAUME_UNIS=Image.open('./image/ROYAUME_UNIS.png')
ITALIE=Image.open('./image/italie.png')
ESPAGNE=Image.open('./image/espagne.png')
NORVEGE=Image.open('./image/norvege.png')
PORTUGAL=Image.open('./image/portugal.png')
CHINE=Image.open('./image/chine.png')
ALLEMAGNE=Image.open('./image/allemagne.png')
BELGIQUE=Image.open('./image/belgique.png')
INDE= Image.open('./image/inde.png')
AUSTRALIE= Image.open('./image/australie.png')
BLANC = Image.open('./image/Blanc.png')

imgP = Image.open('./image/podium.jpg')
```

```
# On redimensionne la taille de l'image du podium ainsi que des drapeaux

height = imgP.size[0]
width = imgP.size[1]

width = width + 1500
height = height + 1200

imgP = imgP.resize((width ,height))

CANADA = CANADA.resize((480 ,437))
ETATS_UNIS = ETATS_UNIS.resize((480 ,437))
ROYAUME_UNIS = ROYAUME_UNIS.resize((480 ,437))
ITALIE = ITALIE.resize((480 ,437))
ESPAGNE = ESPAGNE.resize((480 ,437))
NORVEGE = NORVEGE.resize((480 ,437))
PORTUGAL = PORTUGAL.resize((480 ,437))
CHINE = CHINE.resize((480 ,437))
ALLEMAGNE = ALLEMAGNE.resize((480 ,437))
BELGIQUE = BELGIQUE.resize((480 ,437))
INDE = INDE.resize((480 ,437))
AUSTRALIE = AUSTRALIE.resize((480 ,437))
BLANC = BLANC.resize((480 ,437))

dfo.replace({"ETATS-UNIS": "ETATS_UNIS", "ROYAUME-UNI": "ROYAUME_UNIS"}, inplace=(True))



if str(dfo["nom_entreprise"][0]) in listePaysPodium:
    pou1 = str(dfo["nom_entreprise"][0])
else:
    pou1 = "BLANC"
if str(dfo["nom_entreprise"][1]) in listePaysPodium:
    pou2 = str(dfo["nom_entreprise"][1])
else:
    pou2 = "BLANC"
if str(dfo["nom_entreprise"][2]) in listePaysPodium:
    pou3 = str(dfo["nom_entreprise"][2])
else:
    pou3 = "BLANC"
```

```
position1 = pou1
position2 = pou2
position3 = pou3


for podiumL in listePaysPodium:
    if position2 in listePaysPodium:
        if position2 == podiumL:
            position2=Image.open("" + str(position2) + ".png")
            position2 = position2.resize((480 ,437))
            imgP.paste(position2, (180,630))
    # else:
    #     imgP.text((0, 0), "", fill=(0, 0, 0), font=myFont)

    if position1 in listePaysPodium:
        if position1 == podiumL:
            position1=Image.open("" + str(position1) + ".png")
            position1 = position1.resize((480 ,437))
            imgP.paste(position1, (730,330))
        # else:
        #       imgP.text((0, 0), "", fill=(0, 0, 0), font=myFont)


    if position3 in listePaysPodium:
        if position3 == podiumL:
            position3=Image.open("" + str(position3) + ".png")
            position3 = position3.resize((480 ,437))
            imgP.paste(position3, (1270,800))
        # else:
        #     imgP.text((0, 0), "", fill=(0, 0, 0), font=myFont)




# imgP.paste(position2, (180,300))
# imgP.paste(position1, (730,100))
# imgP.paste(position3, (1350,400))

#imgP.show()


imgP.save("./image/car3.png")
```

Pour la partie positionnement des graphiques et des images sur la page web tout a été réalisé en HTML et CSS. Nous espérons que ce manuel technique vous a permis d'un peu mieux comprendre nos démarches pour la réalisation de ce projet et que vous n'aurez aucun problème à l'utiliser dans le cas où vous voudriez alimenter la base de données AREXIS.