P2-DataSystEd-2_0

August 13, 2019

1 Projet 2 : Analyse des données de systèmes éducatifs

1.1 0. Importations des modules et des données

In [1]:
```
## IMPORTATIONS
import pandas as pd
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

idx  = pd.IndexSlice

import string
import numpy as np
import matplotlib.pyplot as plt
% matplotlib inline
import seaborn as sns
sns.set_style("whitegrid")
#% matplotlib notebook
# graphes interactifs
import re
from wordcloud import WordCloud, STOPWORDS
from collections import Counter
from IPython.display import Image
import scipy.stats as st
#import statsmodels.api as sm
#from sklearn.datasets import load_iris
#iris_df_ori = load_iris()

####     A ESSAYER      ############################################################
# GRAPHES INTERACTIFS


# POUR LES GRAPHIQUES
# % matplotlib inline
# plt.rcParams['figure.figsize'] = [9.5, 6] # ajuster la taille
```

```
        # POUR DESACTIVER LA TOOLBOX GRAPHES TOP GRANDS
        # %%javascript
        # IPython.OutputArea.prototype._should_scroll = function(lines) {
        #     return false;
        # }
```

In [2]:
```
## FONCTION SORTANT UN DATAFRAME D'INFOS (complémentaire du describe)

def desc_bis (df):
    nb_li = df.index.size
    nb_col = df.columns.size
    tot = nb_li*nb_col
    infos = pd.DataFrame(df.dtypes).T.rename(index={0:'Type'})
    infos = infos.append(pd.DataFrame(df.isna().sum()).T.rename(index={0:'null'}))
    return infos
```

In [3]:
```
def infos (df):
    nb_li = df.shape[0]
    nb_co = df.shape[1]
    t = np.empty(nb_li)
    t.fill(nb_li)
    df_l_null = pd.DataFrame(df.T.isna().sum()) # tableau du nbe de nul par lignes (+8
    df_c_null = pd.DataFrame(df.isna().sum()) # tableau du nbe de nul par colonnes (+6

    # nbe de lignes sans 'null'
    al = len([x for x in df_l_null[0] if x==0])
    nb_ss_null = pd.DataFrame([al]).rename(index={0:'lign_ss_null'}).T
    pct_ss_null = pd.DataFrame([al*100/nb_li]).rename(index={0:'lign_ss_null'}).T
    # nbe de lignes 'null'
    bl = len([x for x in df_l_null[0] if x==nb_co])
    nb_null = pd.DataFrame([bl]).rename(index={0:'lign_null'}).T
    pct_null = pd.DataFrame([bl*100/nb_li]).rename(index={0:'lign_null'}).T
    # nbe de lignes mixtes
    cl = len([x for x in df_l_null[0] if (x!=0 and x!=nb_co)])
    nb_mix = pd.DataFrame([cl]).rename(index={0:'lign_mix'}).T
    pct_mix = pd.DataFrame([cl*100/nb_li]).rename(index={0:'lign_mix'}).T
    infos_nb = pd.concat([nb_ss_null, nb_null, nb_mix],axis=1, sort=False).rename(inde
    infos_pct = pd.concat([pct_ss_null, pct_null, pct_mix],axis=1, sort=False).rename(
    infos_l = pd.concat([infos_nb,infos_pct], sort=False)
    # nbe de lignes total
    infos_l["lign_tot"] = [infos_l.T['nb'].sum(), infos_l.T['pct'].sum()]

    # nbe de colonnes sans 'null'
    ac = len([x for x in df_c_null[0] if x==0])
    nb_ss_null = pd.DataFrame([ac]).rename(index={0:'col_ss_null'}).T
    pct_ss_null = pd.DataFrame([ac*100/nb_co]).rename(index={0:'col_ss_null'}).T
    # nbe de colonnes 'null'
    bc = len([x for x in df_c_null[0] if x==nb_li])
```

```
            nb_null = pd.DataFrame([bc]).rename(index={0:'col_null'}).T
            pct_null = pd.DataFrame([bc*100/nb_co]).rename(index={0:'col_null'}).T
            # nbe de colonnes mixtes
            cc = len([x for x in df_c_null[0] if (x!=0 and x!=nb_li)])
            nb_mix = pd.DataFrame([cc]).rename(index={0:'col_mix'}).T
            pct_mix = pd.DataFrame([cc*100/nb_co]).rename(index={0:'col_mix'}).T
            infos_nb = pd.concat([nb_ss_null, nb_null, nb_mix],axis=1, sort=False).rename(index
            infos_pct = pd.concat([pct_ss_null, pct_null, pct_mix],axis=1, sort=False).rename(i
            infos_c = pd.concat([infos_nb,infos_pct], sort=False)
            # nbe de lignes total
            infos_c["col_tot"] = [infos_c.T['nb'].sum(), infos_c.T['pct'].sum()]

            infos = pd.concat([infos_l,infos_c], axis=1, sort=False)

            return infos

In [4]: ## FONCTION DE COMPTAGE DES VALEURS NULLES
        def evalNull (inf_df):
            a = inf_df.T['null'].sum()
            b = inf_df.T['count'].sum()
            print("Nbe valeurs 'null' : {:.0f}".format(a))
            print("Nbe valeurs non 'null' : {:.0f}".format(b))
            print("Nbe total cases : {:.0f}".format(a+b))
            print("% total valeurs 'null' : {:.1f}%".format(a*100/(a+b)))

In [5]: # Fonction qui trouve les éléments uniques différents dans deux tableaux
        def Diff(tab1, tab2):
            #tab_dif = [i for i in tab1 + tab2 if i not in tab1 or i not in tab2] # renvoie en
            return (set(tab1)-set(tab2),set(tab2)-set(tab1)) # renvoie deux tableaux spécifiqu

In [6]: # Fonction vérifiant l'unicité des lignes d'une liste de listes
        def uniCle (t_tab):
            if isinstance(t_tab[0], type(str)) :
                uni = list(set(t_tab))
                res = True if (len(uni)==len(t_tab)) else False
            else :
                uni = list(set(zip(*t_tab))) # liste des combinaisons uniques
                res = True if (len(uni)==len(t_tab[0])) else False
            return res

In [7]: # Fonction comparant la correspondance unique entre les valeurs d'une même ligne de de
        # (bijection entre les valeurs de col1 et de col2)
        def Adeq (df, nom_col1,nom_col2):
            mon_zip = zip(df[nom_col1], df[nom_col2]) # associe les entrées des deux colonnes
            nbe_comb = len(set(mon_zip)) # retourne les valeurs uniques des tuples
            return nbe_comb==df[nom_col1].unique().size # si le nbe est le même que les valeur

In [8]: def recursive_len(item):
            if type(item) == list:
```

```
            return sum(recursive_len(subitem) for subitem in item)
        else:
            return 1
```

In [9]:
```
def contAny (cars, mot):
    return any([True if c in cars else False for c in mot])

def contAll (cars, mot):
    return all([True if c in cars else False for c in mot])

def enum_mots_cmpt(li_phrases, nb): # prend une liste de phrases en entrée
    li_mots = " ".join(li_phrases).split(" ")
    li_mots_net = sorted([mot for mot in li_mots if (mot != '') \
                          and not contAll('-)%.(,', mot)])
    cpt = Counter(li_mots_net)
    words_occ = cpt.most_common(nb) # tableau de tuples
    words = [words_occ[i][0] for i in range(len(words_occ))]
    occs = [words_occ[i][1] for i in range(len(words_occ))]
    dic_occs = dict()
    for i in range(len(words_occ)):
        dic_occs[words[i]]=occs[i]
    return dic_occs # dictionnaire

def filt_dict(dic_t, li_pop):
    dic = dic_t
    [dic.pop(w) for w in li_pop if w in dic_t.keys()]
    return dic
```

In [10]:
```
def random_color_func(word=None, font_size=None, position=None, orientation=None, font
    h = int(360.0 * tone / 255.0)
    s = int(100.0 * 255.0 / 255.0)
    l = int(100.0 * float(np.random.randint(70, 120) / 255.0))
    return "hsl({}, {}%, {}%)".format(h, s, l)
tone = 10.0 # define the color of the words

def nuageMots(dic_occs): # prend un dictionnaire {"mot" : nbe occurences}
    fig = plt.figure(figsize=(18,8))
    wordcloud = WordCloud(width=1000,height=200, background_color='black', max_words=
                          relative_scaling=1, color_func = None, normalize_plurals=False)
    wordcloud.generate_from_frequencies(dic_occs)
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis('off')
    plt.show()

def histMots(dic_occs): # prend un dictionnaire {"mot" : nbe occurences}
    fig = plt.figure(figsize=(18,4))
    tab_occs = np.array([[k,int(v)] for k,v in dic_occs.items()]).T   # dictionnaire
    x = tab_occs[0]
```

```
        y = tab_occs[1].astype(int)
        x_label = tab_occs[0]
        ax = plt.bar(x, y, align = 'center', color='b')
        plt.xticks(x, x_label, rotation=85, fontsize = 15)
        plt.yticks(fontsize = 15)
        plt.ylabel("Nb. of occurences", fontsize = 18, labelpad = 10)
        plt.title("Fréquence des mots-clés",color='k',fontsize = 18, fontweight = 'bold')
        plt.show() # affiche l'histogramme
```

```
In [11]: ## FONCTION D'AFFICHAGE
         def basic_plot(plot_type, my_plot, my_x, my_y, my_x_t, xlab, ylab, my_tit, num_col):
             if plot_type == "plot" :
                 my_plot.plot(my_x, my_y, '-o', color = colors[num_col])
             elif plot_type == "bar" :
                 my_plot.bar(my_x, my_y, color = colors[num_col])
             elif plot_type == "scatter" :
                 my_plot.scatter(my_x, my_y, color = colors[num_col])
             else :
                 print("erreur type de graphe")

             my_plot.set_xlabel(xlab, fontsize = 14)
             my_plot.set_ylabel(ylab, fontsize = 14)
             if my_x_t == '':
                 plt.xticks(fontsize = 14)
             else :
                 plt.xticks(my_x_t, my_x_t, rotation=85 , fontsize = 14)
             plt.yticks(fontsize = 14)
             plt.ylim(round(min(my_y)*0.9), round(max(my_y)*1.1))
             my_plot.set_title(my_tit, fontsize = 18, fontweight = 'bold')
             plt.grid(color='grey', linestyle='dotted')
```

L'ensemble des données téléchargées se compose de 5 fichiers .csv et d'un fichier excel comportant 5 onglets. Il semble que l'intégralité des données des fichiers .csv soit reprise dans chacun des onglets du fichier Excel.

Dans ce notebook, on appellera "base de donnée" l'ensemble des données, et "table" chacun des onglets ou fichier .csv correspondant.

On travaillera sur les cinq dataframes créées dans la cellule suivante :

```
In [12]: # Utilisé la fonction dropna (colonne nulle) pour éliminer les colonnes fantômes "Una
         data = pd.read_csv("../DONNEES/EdStatsData.csv").dropna(how='all', axis='columns')
         country = pd.read_csv("../DONNEES/EdStatsCountry.csv").dropna(how='all', axis='columns
         cnt_ser = pd.read_csv("../DONNEES/EdStatsCountry-Series.csv").dropna(how='all', axis=
         series = pd.read_csv("../DONNEES/EdStatsSeries.csv").dropna(how='all', axis='columns'
         footnote = pd.read_csv("../DONNEES/EdStatsFootNote.csv").dropna(how='all', axis='colu
```

1.2 1. Vérification et rectification de la qualité des données

On crée 5 nouvelles dataframes "data_c", "country_c", "series_c", "cnt_ser_c" et "footnote_c" qui contiendront les données rectifiées :

In [13]:
```
# on crée d'autres dataframes à modifier (deep copies)
data_c = data.copy()
country_c = country.copy()
series_c = series.copy()
cnt_ser_c = cnt_ser.copy()
footnote_c = footnote.copy()
```

1.2.1 1.0 Description globale des tables

Comptage des 'null' par lignes et par colonnes pour toutes les tables

In [14]:
```
## COMPTAGE DES 'NULL' par LIGNES et par COLONNES (toutes les tables)
infos_t = pd.concat([infos(data), infos(country), infos(series),\
                     infos(cnt_ser), infos(footnote)], axis = 0,\
                    keys=['data', 'country', 'series', 'cnt_ser', 'footnote'])
pd.options.display.float_format = '{:.1f}'.format
infos_t
```

Out[14]:

		lign_ss_null	lign_null	lign_mix	lign_tot	col_ss_null	col_null	col
data	nb	0.0	0.0	886930.0	886930.0	4.0	0.0	
	pct	0.0	0.0	100.0	100.0	5.8	0.0	
country	nb	0.0	0.0	241.0	241.0	4.0	0.0	
	pct	0.0	0.0	100.0	100.0	12.9	0.0	
series	nb	0.0	0.0	3665.0	3665.0	5.0	0.0	
	pct	0.0	0.0	100.0	100.0	33.3	0.0	
cnt_ser	nb	613.0	0.0	0.0	613.0	3.0	0.0	
	pct	100.0	0.0	0.0	100.0	100.0	0.0	
footnote	nb	643638.0	0.0	0.0	643638.0	4.0	0.0	
	pct	100.0	0.0	0.0	100.0	100.0	0.0	

Table "Data"

In [15]:
```
inf_data = desc_bis(data).append(data.describe(include='all'))
```

In [16]:
```
evalNull(inf_data)
inf_data
```

```
Nbe valeurs 'null' : 52568249
Nbe valeurs non 'null' : 8629921
Nbe total cases : 61198170
% total valeurs 'null' : 85.9%
```

Out[16]:

	Country Name	Country Code	Indicator Name
Type	object	object	object
null	0	0	0
count	886930	886930	886930
unique	242	242	3665
top	Nicaragua	PNG	Wittgenstein Projection: Percentage of the pop...

freq	3665	3665	242
mean	NaN	NaN	NaN
std	NaN	NaN	NaN
min	NaN	NaN	NaN
25%	NaN	NaN	NaN
50%	NaN	NaN	NaN
75%	NaN	NaN	NaN
max	NaN	NaN	NaN

	2060	2065	2070	2075	2080	2085	2090	2095
Type	float64	float64	float64	float64	float64	float64	float64	float64
null	835494	835494	835494	835494	835494	835494	835494	835494
count	51436.0	51436.0	51436.0	51436.0	51436.0	51436.0	51436.0	51436.0
unique	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	722.5	727.1	728.4	726.6	722.8	717.7	711.3	703.4
std	22158.4	22879.9	23523.4	24081.5	24559.0	24965.9	25301.8	25560.7
min	-1.6	-1.4	-1.3	-1.1	-0.9	-0.8	-0.7	-0.6
25%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50%	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
75%	7.5	7.5	7.3	7.1	6.7	6.1	5.5	4.7
max	2951568.8	3070878.8	3169710.6	3246239.2	3301586.2	3337871.2	3354746.3	3351886.9

Table "Country"

In [17]:
```
inf_country = desc_bis(country).append(country.describe(include='all'))
```

In [18]:
```
evalNull(inf_country)
inf_country
```

```
Nbe valeurs 'null' : 2113
Nbe valeurs non 'null' : 5358
Nbe total cases : 7471
% total valeurs 'null' : 28.3%
```

Out[18]:

	Country Code	Short Name	Table Name	Long Name	2-alpha code	Currency Unit	
Type	object	object	object	object	object	object	
null	0	0	0	0	3	26	
count	241	241	241	241	238	215	
unique	241	241	241	241	238	152	
top	MAF	El Salvador	El Salvador	Hungary	MY	Euro	Ap
freq	1	1	1	1	1	23	
mean	NaN	NaN	NaN	NaN	NaN	NaN	
std	NaN	NaN	NaN	NaN	NaN	NaN	
min	NaN	NaN	NaN	NaN	NaN	NaN	
25%	NaN	NaN	NaN	NaN	NaN	NaN	

```
50%             NaN          NaN          NaN       NaN          NaN          NaN
75%             NaN          NaN          NaN       NaN          NaN          NaN
max             NaN          NaN          NaN       NaN          NaN          NaN
```

Table "Series"

```
In [19]: inf_series = desc_bis(series).append(series.describe(include='all'))

In [20]: evalNull(inf_series)
         inf_series

Nbe valeurs 'null' : 33213
Nbe valeurs non 'null' : 21762
Nbe total cases : 54975
% total valeurs 'null' : 60.4%


Out[20]:              Series Code             Topic                                    Indi
         Type              object            object
         null                   0                 0
         count               3665              3665
         unique              3665                37
         top     PRJ.ATT.2024.1.MF  Learning Outcomes  Repetition rate in Grade 5 of primary ed
         freq                   1              1046
```

Table "CountrySeries"

```
In [21]: inf_cnt_ser = desc_bis(cnt_ser).append(cnt_ser.describe(include='all'))

In [22]: evalNull(inf_cnt_ser)
         inf_cnt_ser

Nbe valeurs 'null' : 0
Nbe valeurs non 'null' : 1839
Nbe total cases : 1839
% total valeurs 'null' : 0.0%


Out[22]:        CountryCode   SeriesCode                                      DESCRIPTION
         Type        object       object                                           object
         null             0            0                                                0
         count          613          613                                              613
         unique         211           21                                               97
         top            GEO  SP.POP.GROW  Data sources : United Nations World Population...
         freq            18          211                                              154
```

Table "FootNote"

```
In [23]: inf_footnote = desc_bis(footnote).append(footnote.describe(include='all'))

In [24]: evalNull(inf_footnote)
         inf_footnote

Nbe valeurs 'null' : 0
Nbe valeurs non 'null' : 2574552
Nbe total cases : 2574552
% total valeurs 'null' : 0.0%
```

Out[24]:

	CountryCode	SeriesCode	Year	DESCRIPTION
Type	object	object	object	object
null	0	0	0	0
count	643638	643638	643638	643638
unique	239	1558	56	9102
top	LIC	SH.DYN.MORT	YR2004	Country Data
freq	7320	9226	27128	191188

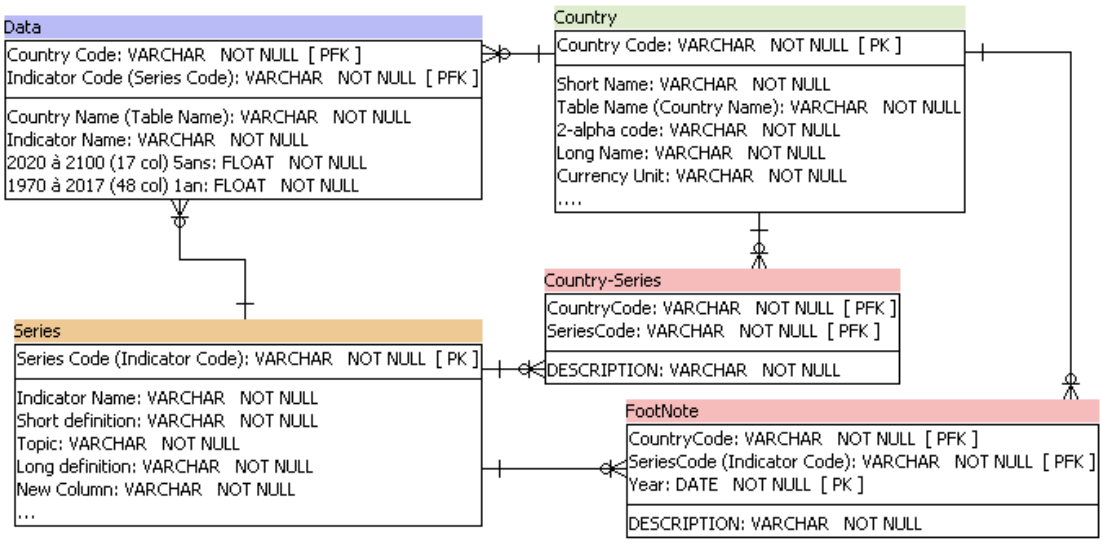
1.2.2 ---------- Bilan description globale ----------

1. La table "Data" est composée de 886930 lignes qui correspondent à toutes les combinaisons des entrées uniques des colonnes "Country Code" et "Indicator Code" (respectivement 242 et 3665 entrées uniques). On trouve dans chaque ligne correspondant à un couple Indicateur/Pays la valeur d'un indicateur pour un pays. Les colonnes détaillent les valeurs pour chaque année de 1970 à 2017 (48 colonnes), puis des projections de ces valeurs tous les 5 ans des années 2020 à 2100 (17 colonnes)
2. La table "Country" est composée de 241 lignes correspondant aux pays sur lesquels portent les données. (Il manque un pays, voir plus bas). Chaque colonne donne des renseignements sur les pays.
3. La table "Series" est composée de 3665 lignes correspondant chacune à un indicateur statistique.
4. La table "Country-series" comporte 613 lignes donne des indications sur les sources des données de divers couples Indicateur/Pays.
5. La table "FootNote" comporte 643638 lignes, et donne des précisions (mode de calcul ou autre) relatives à divers couples Indicateur/Pays.

L'analyse de la composition des tables a permis de déduire le MPD (Modèle Physique de Données) de la base représenté ci-dessous :

```
In [25]: Image("../UML/EduStatsMPD.png")
```

Out[25]:



Les clés proposées dans le MPD ci-dessus pour chaque jeu faciliteraient l'utilisation de la base, sous réserve de quelques modifications (voir modifications plus bas) : - renommer certaines colonnes afin de lever les ambiguïtés (ex : Series/"Series Code" -> Series/"Indicator Code") - supprimer les colonnes redondantes ou inutiles (ex : Data/"Country Name", accessibles via la clé étrangère "Country Code" en colonne Country/"Table Name")

1.2.3 1.1 Données dupliquées ou contradictoires

Vérification de l'unicité des clés de chaque table La description ci-dessus nous a permis de déterminer les colonnes de chaque bases susceptibles de jouer le rôle de clés. Afin de faciliter l'exploitation de la base, il est important que la clé de chaque table soit unique. Vérifions donc que les tables sont exemptes de doublons :

```
In [26]: print("Table Data : Unicité de la clé ('Country Code'&'Indicator Code') {}"\
               .format(uniCle([data["Country Code"], data["Indicator Code"]])))
         print("Table Country : Unicité de la clé ('Country Code') {}"\
               .format(uniCle(country["Country Code"])))
         print("Table Series : Unicité de la clé ('Series Code') {}"\
               .format(uniCle(series["Series Code"])))
         print("Table Country-Series : Unicité de la clé ('CountryCode'&'SeriesCode') {}"\
               .format(uniCle([cnt_ser["CountryCode"], cnt_ser["SeriesCode"]])))
         print("Table FootNote : Unicité de la clé ('CountryCode'&'SeriesCode'&'Year') {}"\
               .format(uniCle([footnote["CountryCode"],footnote["SeriesCode"],footnote["Year"]]
```

```
Table Data : Unicité de la clé ('Country Code'&'Indicator Code') True
Table Country : Unicité de la clé ('Country Code') True
Table Series : Unicité de la clé ('Series Code') False
Table Country-Series : Unicité de la clé ('CountryCode'&'SeriesCode') True
Table FootNote : Unicité de la clé ('CountryCode'&'SeriesCode'&'Year') True
```

Les tables sont bien exemptes de doublons. Pour plus de clarté, on renomme les colonnes contenant des entrées apparentées :

```
In [27]: country_c.rename(columns={'Table Name':'Country Name'}, inplace=True)
         series_c.rename(columns={'Series Code':'Indicator Code'}, inplace=True)
         cnt_ser_c.rename(columns={'CountryCode':'Country Code',
                                   'SeriesCode':'Indicator Code',
                                  'DESCRIPTION':'Desc Data'}, inplace=True)
         footnote_c.rename(columns={'CountryCode':'Country Code',
                                   'SeriesCode':'Indicator Code',
                                  'DESCRIPTION':'Footnote Data'}, inplace=True)
```

Correspondance des entrées uniques des colonnes apparentées

- Les tables "Data" et "Country" n'ont pas le même nombre d'entrée uniques pour la colonne "Country Code" (voir dataframes inf_data et inf_country) : la table "Country" contient un pays en moins. On cherche à déterminer ce pays :

```
In [28]: cnt_lack = Diff(list(data["Country Code"].unique()),list(country["Country Code"].uniq
         print("Codes des pays spécifiques à 'Data' : {}"\
           .format(cnt_lack[0]))
         print("Codes des pays spécifiques à 'Country' : {}"\
           .format(cnt_lack[1]))
         code = list(cnt_lack[0])[0]
         nom = list(data[data["Country Code"]==(list(cnt_lack[0])[0])]["Country Name"].unique()
         print("Code et nom du pays à ajouter : {},{}".format(code, nom))
```

```
Codes des pays spécifiques à 'Data' : {'VGB'}
Codes des pays spécifiques à 'Country' : set()
Code et nom du pays à ajouter : VGB,British Virgin Islands
```

La table Country étant supposée renseigner sur tous les pays présents dans la base, on rajoute une ligne dans la base "Country" contenant le pays manquant :

```
In [29]: line_vgb = pd.DataFrame([[code]+[None]+[nom]+[None]*(country_c.columns.size-3)],\
                                  columns = country_c.columns) # ligne à ajouer
         country_c = country_c.append(line_vgb,'sort=False') # ajout en bas de la liste
         country_c = country_c.sort_values("Country Code") # remet les lignes en ordre alphabé
         country_c.index = list(np.arange(country_c.index.size)) # renumérote l'index

In [30]: #country_c.drop_duplicates(subset='Country Name', keep='first', inplace=True)
```

- Les entrées uniques des colonnes "Country Name" et "Table Name" des tables "Data" et "Country" ne correspondent pas :

```
In [31]: cnt_lack = Diff(list(data_c["Country Name"].unique()),list(country_c["Country Name"].u

         print("* Noms des pays spécifiques à 'Data' : {}"\
```

```
        .format(sorted(cnt_lack[0])))
print("----------------")
print("* Noms des pays spécifiques à 'Country' : {}"\
        .format(sorted(cnt_lack[1])))
print("----------------")
print("* Nombre de pays non concordants : {}, {}".format(len(cnt_lack[0]), len(cnt_lac
# liste complète des codes des pays posant problème (data, puis country) :
l_data = sorted([data_c[data_c["Country Name"]==nom_col].iloc[0]["Country Code"] for n
l_country = sorted([country_c[country_c["Country Name"]==nom_col].iloc[0]["Country Co
l_pbe = sorted(list(set(l_data + l_country)))
print("----------------")
print("* Liste des codes des {} pays posant problème : {}".format(len(l_pbe), l_pbe))
```

```
* Noms des pays spécifiques à 'Data' : ["Cote d'Ivoire", 'Curacao', 'East Asia & Pacific (exclu
----------------
* Noms des pays spécifiques à 'Country' : ['Curaçao', "Côte d'Ivoire", 'East Asia & Pacific (a
----------------
* Nombre de pays non concordants : 10, 10
----------------
* Liste des codes des 15 pays posant problème : ['CIV', 'CUW', 'EAP', 'EAS', 'ECA', 'ECS', 'FR
```

In [32]:
```
# tableau comparatif des noms de pays 'posant problème' et ayant le même code dans da
mask1 = [li.any() for li in np.array([np.array((data["Country Code"]==n).values) for n
mask2 = [li.any() for li in np.array([np.array((country["Country Code"]==n).values) fo
comp = pd.merge(data[mask1], country[mask2], left_on= "Country Code", right_on= "Coun
tab_ser = [comp[comp["Country Code"] == pays].iloc[0][['Country Code','Country Name',
           for pays in l_pbe]
my_df = pd.DataFrame(tab_ser, columns = ['Country Code','Country Name', 'Table Name']
my_df.columns = ['Country Code','Country Name (data_c)', 'Country Name (country_c)' ]
my_df.sort_values(by = ['Country Name (country_c)'], ascending = True, inplace = True
my_df = my_df.reset_index(drop=True)
my_df.head()
```

Out[32]:

	Country Code	Country Name (data_c)	Country
0	CUW	Curacao	
1	CIV	Cote d'Ivoire	
2	EAP	East Asia & Pacific (excluding high income)	Eas
3	EAS	East Asia & Pacific	East Asia & Pacific (a
4	ECA	Europe & Central Asia (excluding high income)	Euro

On conserve les noms de la table 'Data', qui ne contiennent pas de caractères spéciaux, et qui sont plus clairs sur la désignation ('excluding high income' préférable à 'all income levels') :

In [33]:
```
country_c["Country Name"].replace(to_replace = my_df['Country Name (country_c)'].value
                                  value = my_df['Country Name (data_c)'].values, inplace=True)
```

- Les tables "Data" et "Series" ont bien le même nombre d'entrée uniques pour la colonne "Indicator Code" et "Series Code" (voir dataframes inf_data et inf_country) : 3665. Vérifions si ces entrées uniques sont bien les mêmes :

```
In [34]: serie_lack = Diff(list(data_c["Indicator Code"].unique()),list(series_c["Indicator Co

         print("Nbe d'indicateurs spécifiques à 'Data' : {}"\
           .format(len(serie_lack[0])))
         print("Nbe d'indicateurs spécifiques à 'Series' : {}"\
           .format(len(serie_lack[1])))
         print("Quelques indicateurs spécifiques à 'Data' :\n {}"\
           .format(sorted(serie_lack[0])[:5]))
         print("Quelques indicateurs spécifiques à 'Series' :\n {}"\
           .format(sorted(serie_lack[1])[:5]))

Nbe d'indicateurs spécifiques à 'Data' : 53
Nbe d'indicateurs spécifiques à 'Series' : 53
Quelques indicateurs spécifiques à 'Data' :
 ['SE.SEC.DURS.LO', 'SE.SEC.ENRR.UP.FE', 'UIS.AIR.1.GLAST.GPI', 'UIS.CEAGE.1', 'UIS.E.0.PU.F']
Quelques indicateurs spécifiques à 'Series' :
 ['SE.SEC.DURS.LO ', 'SE.SEC.ENRR.UP.FE ', 'UIS.AIR.1.Glast.GPI', 'UIS.CEAge.1', 'UIS.E.0.Pu.F
```

L'échantillon d'indicateurs affichés ci-dessus ne diffèrent en fait que par 1) des espaces 2) des lettres en minuscule dans la table "Series". Vérifions qu'après correction les entrées sont les mêmes :

```
In [35]: test_ser = pd.Series([series["Series Code"][i].upper().replace(" ", "")\
                               for i in range(series["Series Code"].index.size)],\
                                index = series["Series Code"].index)
         test_data = pd.Series([data["Indicator Code"][i].upper().replace(" ", "")\
                               for i in range(data["Indicator Code"].index.size)],\
                                index = data["Indicator Code"].index)
         lack1 = Diff(list(data["Indicator Code"].unique()),list(test_ser.unique()))
         lack2 = Diff(list(test_data.unique()), list(series["Series Code"].unique()))
         lack3 = Diff(list(test_data.unique()), list(test_ser.unique()))

         print("-----Correction de 'Series' seule :-----")
         print("Nbe d'indicateurs spécifiques à 'Data', puis 'Series' : {}, {}"\
           .format(len(lack1[0]), len(lack1[1])))
         print("-----Correction de 'Data' seule :-----")
         print("Nbe d'indicateurs spécifiques à 'Data', puis 'Series' : {}, {}"\
           .format(len(lack2[0]), len(lack2[1])))
         print("-----Correction de 'Data' et 'Series' :-----")
         print("Nbe d'indicateurs spécifiques à 'Data', puis 'Series' : {}, {}"\
           .format(len(lack3[0]), len(lack3[1])))

-----Correction de 'Series' seule :-----
Nbe d'indicateurs spécifiques à 'Data', puis 'Series' : 5, 5
-----Correction de 'Data' seule :-----
Nbe d'indicateurs spécifiques à 'Data', puis 'Series' : 58, 58
-----Correction de 'Data' et 'Series' :-----
```

```
Nbe d'indicateurs spécifiques à 'Data', puis 'Series' : 0, 0
```

La correction des noms des deux bases est nécessaire. On effectue la correction des indicateurs dans la base 'Data', dans la base 'Series'.

```
In [36]: data_c["Indicator Code"].replace(to_replace = serie_lack[0], \
                                  value = [x.upper().replace(" ", "") for x in serie_lack[0]], i
         series_c["Indicator Code"].replace(to_replace = serie_lack[1], \
                                       value =[x.upper().replace(" ", "") for x in serie_lack[1]

In [37]: serie_lack2 = Diff(list(data_c["Indicator Code"].unique()),list(series_c["Indicator Co
         print("Nbe d'indicateurs spécifiques à 'Data' et à 'Series' après modification : {}, {
               .format(len(serie_lack2[0]),len(serie_lack2[1])))
```

```
Nbe d'indicateurs spécifiques à 'Data' et à 'Series' après modification : 0, 0
```

- Vérifions que les codes d'indicateurs et de pays présents dans les tables "Country-Series" et "Footnote" sont bien dans la liste des codes d'indicateurs de la table "Series" et dans la liste des codes de pays de la table "Country" :

```
In [38]: cnt_ser_lack1 = Diff(list(series_c["Indicator Code"].unique()),list(cnt_ser_c["Indicat
         cnt_ser_lack2 = Diff(list(country_c["Country Code"].unique()),list(cnt_ser_c["Country
         footnote_lack1 = Diff(list(series_c["Indicator Code"].unique()),list(footnote_c["Indi
         footnote_lack2 = Diff(list(country_c["Country Code"].unique()),list(footnote_c["Countr

         print("-------- table 'CountrySeries' --------")
         print("Nbe d'indicateurs/de pays spécifiques à 'Series'/'Country' : {}/{}"\
           .format(len(cnt_ser_lack1[0]), len(cnt_ser_lack2[0])))
         print("Nbe d'indicateurs/de pays spécifiques à 'CountrySeries' : {}/{}"\
           .format(len(cnt_ser_lack1[1]), len(cnt_ser_lack2[1])))
         print("-------- table 'FootNote' --------")
         print("Nbe d'indicateurs/de pays spécifiques à 'Series'/'Country' : {}/{}"\
           .format(len(footnote_lack1[0]), len(footnote_lack2[0])))
         print("Nbe d'indicateurs/de pays spécifiques à 'FootNote' : {}/{}"\
           .format(len(footnote_lack1[1]), len(footnote_lack2[1])))
```

```
-------- table 'CountrySeries' --------
Nbe d'indicateurs/de pays spécifiques à 'Series'/'Country' : 3644/31
Nbe d'indicateurs/de pays spécifiques à 'CountrySeries' : 0/0
-------- table 'FootNote' --------
Nbe d'indicateurs/de pays spécifiques à 'Series'/'Country' : 2196/3
Nbe d'indicateurs/de pays spécifiques à 'FootNote' : 89/0
```

La base 'CountrySeries' ne contient ni de code de pays ni de code d'indicateur qui ne soit pas dans les bases 'Country' et 'Series', en revanche, la base 'FootNote' contient des codes d'indicateurs erronnés.

```
In [39]: print("-------------")
         # pays pas dans footnote,
         #print("Quelques code pays seulement dans footnote :\n{}".format(list(footnote_lack2[
         #pays seulement dans footnote
         print("Quelques code pays seulement dans footnote :\n{}".format(list(footnote_lack2[1
         print("-------------")
         # codes indicateurs pas dans footnote,
         #print("Quelques indicateurs de 'series' pas dans 'footnote' :\n{}".format(list(footn
         #codes indicateurs seulement dans footnote
         print("Quelques indicateurs seulement dans 'footnote' :\n{}".format(list(footnote_lack
```

```
-------------
Quelques code pays seulement dans footnote :
[]
-------------
Quelques indicateurs seulement dans 'footnote' :
['UIS.LR.Ag15t99.GPI', 'UIS.LPP.Ag15t24', 'UIS.XGovExp.IMF.56', 'UIS.GER.1t6.GPI', 'UIS.XUNIT.
```

Les codes d'indicateurs de 'FootNote' contiennent des lettres en minuscules. On applique la même modification que pour les tables 'Data' et 'Series' précédemment (élimination des espaces, passage en majuscules). On vérifie qu'après correction 'FootNote' n'a pas de codes d'indicateurs spécifiques.

```
In [40]: footnote_c["Indicator Code"].replace(to_replace = footnote_lack1[1], \
                                    value = [x.upper().replace(" ", "") for x in footnote_lack1[1]
         footnote_lack = Diff(list(footnote_c["Indicator Code"].unique()),list(series_c["Indica
         print("Nbe d'indicateurs spécifiques à 'Data' et à 'Series' après modification : {}, 
               .format(len(footnote_lack[0]),len(footnote_lack[1])))
```

```
Nbe d'indicateurs spécifiques à 'Data' et à 'Series' après modification : 0, 2183
```

- Y a-t-il une correspondance entre les noms d'indicateurs de la table "Series" et ceux de la table "Data" ? Si oui, comme pour les noms de pays, on ne gardera qu'une des deux colonnes.

On remarque que 462 entrées uniques de la colonne "Indicator Name" de la table "data" ne sont pas dans la table "series" et que le même nombre 462 d'entrées uniques de la même colonne de la table "series" ne sont pas dans la table "data".

```
In [41]: indic_lack = Diff(list(data_c["Indicator Name"].unique()),list(series["Indicator Name"
         #noms d'indicateurs seulement dans data
         print("Quelques indicateurs seulement dans 'Data' :\n{}".format(list(indic_lack[0])[:
         #noms d'indicateurs seulement dans series
         print("Quelques indicateurs seulement dans 'Series' :\n{}".format(list(indic_lack[1])
         print("----------------")
```

```
        print("* Nombre de pays non concordants : {}, {}".format(len(indic_lack[0]), len(indic
        # liste complète des codes des indicateurs posant problème (data, puis series) :
        l_data = sorted([data_c[data_c["Indicator Name"]==nom_col].iloc[0]["Indicator Code"] :
        l_series = sorted([series_c[series_c["Indicator Name"]==nom_col].iloc[0]["Indicator Co
        l_pbe = sorted(list(set(l_data + l_series)))
        print("----------------")
        print("* Liste des codes de quelques-uns des {} indicateurs posant problème : {}".form
```

```
Quelques indicateurs seulement dans 'Data' :
['Wittgenstein Projection: Percentage of the population age 15+ by highest level of educational
Quelques indicateurs seulement dans 'Series' :
['Age population, age 18, female, UNESCO', 'Age population, age 03, female, UNESCO', 'Projectio
----------------
* Nombre de pays non concordants : 462, 462
----------------
* Liste des codes de quelques-uns des 462 indicateurs posant problème : ['LO.LLECE.MAT3', 'LO.L
```

In [42]:
```
# tableau comparatif des noms de pays 'posant problème' et ayant le même code dans da
mask1 = [li.any() for li in np.array([np.array((data_c["Indicator Code"]==n).values) :
mask2 = [li.any() for li in np.array([np.array((series_c["Indicator Code"]==n).values)
comp = pd.merge(data_c[mask1], series_c[mask2], left_on= "Indicator Code", right_on= '
tab_ser = [comp[comp["Indicator Code"] == ind].iloc[0][['Indicator Code','Indicator Na
           for ind in l_pbe]
```

In [43]:
```
# Affichage des noms complets
# comp[['Indicator Code','Indicator Name_x', 'Indicator Name_y', 'Country Name']]\
# .groupby(['Indicator Code','Indicator Name_x','Indicator Name_y']).count()
```

In [44]:
```
my_df = pd.DataFrame(tab_ser, columns = ['Indicator Code','Indicator Name_x', 'Indicat
my_df.columns = ['Indicator Code','Indicator Name (data_c)', 'Indicator Name (series_
my_df.sort_values(by = ['Indicator Name (series_c)'], ascending = True, inplace = Tru
#my_df = my_df.reset_index(drop=True)
my_df.head()
```

Out[44]:

	Indicator Code	Indicator Name (data_c)	Indicator Name (seri
12342	SP.POP.AG00.FE.UN	Population, age 0, female	Age population, age 0, female, U
12584	SP.POP.AG00.TO.UN	Population, age 0, total	Age population, age 0, total, U
12826	SP.POP.AG01.FE.UN	Population, age 1, female	Age population, age 01, female, U
13068	SP.POP.AG01.TO.UN	Population, age 1, total	Age population, age 01, total, U
18150	SP.POP.AG02.FE.UN	Population, age 2, female	Age population, age 02, female, U

Les noms de la table 'Series' sont généralement plus complets que ceux de 'Data'. On remplace donc les valeurs de 'Indicator Name' dans 'data_c' par celles de la même colonne dans 'series_c' :

In [45]:
```
############# A FAIRE  ##########################
# Remplacement par le nom le plus long des deux
# Vérifier quand même que des indicateurs plus courts n'ont pas été remplacés par des
#################################################"""""""
```

```
In [46]: # remplacement des noms de la colonne data, par ceux de la colonne series
         data_c["Indicator Name"].replace(to_replace = my_df['Indicator Name (data_c)'].values
                                  value = my_df['Indicator Name (series_c)'].values, inplace=Tru

In [47]: #Vérification de la correspondance après modification
         verif = Diff(sorted(data_c["Indicator Name"].unique()), sorted(series_c["Indicator Nam
         list(verif[0])[:5], list(verif[1])[:5]

Out[47]: ([], [])

In [48]: len(data_c["Indicator Name"].unique()), len(series_c["Indicator Name"].unique())

Out[48]: (3665, 3665)
```

Simplification de la base

```
In [49]: ############# A FAIRE  ##########################
         # Fusion des tables footnote_c et cnt_ser_c en une table inf_data_c
         ################################################
```

- Les années de la table "FootNote" sont apparemment dans un format string et précédés de YR ou yr.

```
In [50]: # liste des années de la colonne "Year" de la table "FootNote"
         print(set(footnote_c["Year"].unique()))

{'YR1972', 'YR1987', 'YR2030', 'YR2003', 'YR2015', 'YR2020', 'yr2012', 'YR2002', 'YR2009', 'YR
```

On remplace les valeurs des chaînes par l'entier correspondant :

```
In [51]: # remplacement des années de la colonne 'Year', par les nombres correspondants
         ch_ann_uni = list(footnote_c["Year"].unique())
         int_ann_uni = [int(('').join(re.findall("[0-9]", chaine))) for chaine in ch_ann_uni]
         footnote_c["Year"].replace(to_replace = ch_ann_uni, value = int_ann_uni, inplace=True)

In [52]: print(set(footnote_c["Year"].unique()))

{2050, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984
```

Relations bijectives entre colonnes d'une même base Après vérification de l'unicité des clés choisies pour chaque table, on vérifie la correspondance bijective entre plusieurs paires de colonnes d'une même table, par exemple : - les codes de pays et les noms de pays (dans 'Data' et dans 'Country') - les codes d'indicateurs et les noms d'indicateurs (dans 'Data' et dans 'Series')

On teste (voir définition de la fonction 'Adeq' plus haut) les couples de colonnes listés ci-dessous : - Table "Data" : "Country Code", "Country Name" (Dataframe "data_c" : "Country Code", "Country Name") - Table "Data" : "Indicator Code", "Indicator Name" (Dataframe "data_c" : "Indicator Code", "Indicator Name") - Table "Country" : "Country Code", "Short Name" (Dataframe "country_c" : "Country Code", "Short Name") - Table "Country" : "Country Code", "Table Name" (Dataframe "country_c" : "Country Code", "Country Name") - Table "Country" : "Country Code", "Long Name" (Dataframe "country_c" : "Country Code", "Long Name") - Table "Series" : "Series Code", "Indicator Name" (Dataframe "series_c" : "Indicator Code", "Indicator Name")

```
In [53]: print("Data : {} {}"\
               .format(Adeq(data_c, "Country Code", "Country Name"), Adeq(data_c, "Indicator C
         print("Country : {} {} {}"\
               .format(Adeq(country_c, "Country Code", "Short Name"), Adeq(country_c, "Country
                 Adeq(country_c, "Country Code", "Long Name")))
         print("Series : {}".format(Adeq(series_c, "Indicator Code", "Indicator Name")))
```

```
Data : True True
Country : True True True
Series : True
```

Ajout à la table 'Data' Pour faciliter le traitement ultérieur des données de la table "Data", on ajoute deux colonnes précisant : - la région du pays - et le topic de l'indicateur

```
In [54]: # ajout de la région du pays
         data_c = pd.merge( data_c , country_c[['Country Code', 'Region']] , left_on = 'Country
         # ajout du topic de l'indicateur
         data_c = pd.merge( data_c , series_c[['Indicator Code', 'Topic']] , left_on = 'Indicat
         cols = list(data_c.columns.values)
         cols = cols[-2:]+cols[:-2]
         data_c = data_c[cols] # remise en ordre des colonnes
```

Elimination des colonnes sous-remplies Il existe des colonnes dans les tables 'Country', 'Series', 'Country-Series' et 'Footnote' qui sont très peu remplies. Cependant, les données contenues dans ces tables ne sont pas indispensables au traitement des données chiffrées, qui sont contenues dans la table 'Data'. On n'éliminera donc pas ces colonnes. En ce qui concerne la table 'Data', elle contient seulement deux colonnes peu remplies (années 2016 et 2017) qu'il n'est pas nécessaire d'effacer pour l'instant.

Groupes de pays en régions

- Dans la liste country_c["Country Name"], on remarque 27 pays n'ayant pas de valeur de country["Region"] et country["Income group"]. Ceux-ci sont en fait des groupes de pays. On élimine les données relatives à ces pays (data_c) de notre liste d'intérêt.

```
In [55]: # comptage des pays uniques sans région
         sans_reg_cnt = country_c[country_c["Region"].isna()]["Country Name"]
         li_pays_supp = sans_reg_cnt.values
         #['Arab World', 'East Asia & Pacific (excluding high income)','East Asia & Pacific',
         print(Diff(sans_reg_cnt, sans_reg_cnt) , len(sans_reg_cnt))
         print(li_pays_supp) # faux 'pays' à éliminer
```

```
(set(), set()) 28
['Arab World' 'East Asia & Pacific (excluding high income)'
 'East Asia & Pacific' 'Europe & Central Asia (excluding high income)'
 'Europe & Central Asia' 'Euro area' 'European Union' 'Gibraltar'
 'High income' 'Heavily indebted poor countries (HIPC)'
```

```
 'Latin America & Caribbean (excluding high income)'
 'Latin America & Caribbean'
 'Least developed countries: UN classification' 'Low income'
 'Lower middle income' 'Low & middle income' 'Middle East & North Africa'
 'Middle income' 'Middle East & North Africa (excluding high income)'
 'North America' 'Nauru' 'OECD members' 'South Asia'
 'Sub-Saharan Africa (excluding high income)' 'Sub-Saharan Africa'
 'Upper middle income' 'British Virgin Islands' 'World']
```

```
In [56]: # élimination des faux 'pays' dans "Country"
         ind_supp_cnt = sans_reg_cnt.index # index des lignes du tableau country à éliminer (u
         country_c.drop(index = ind_supp_cnt, inplace = True)
         # élimination des faux 'pays' dans "Data"
         sans_reg_data = [data_c[data_c["Country Name"]== col].index for col in li_pays_supp]
         ind_supp_data = [item for sublist in sans_reg_data for item in sublist] # liste des i
         data_c.drop(index = ind_supp_data, inplace = True)
```

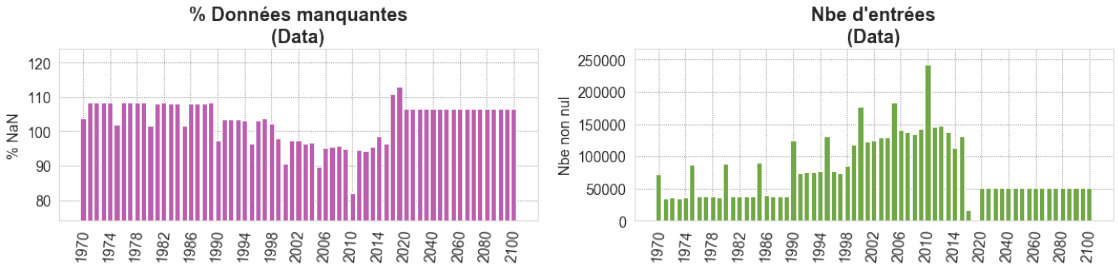
```
In [57]: # vérification
         sans_reg_cnt = country_c[country_c["Region"].isna()]["Country Name"]
         len(sans_reg_cnt)
         sans_reg_cnt = country_c[country_c["Region"].isna()]["Country Name"]
         sans_reg_data = data_c[data_c["Region"].isna()]["Country Name"]
         print(len(sans_reg_cnt))
```

```
0
```

1.2.4 1.2 Données manquantes

Comptage des données manquantes par table

```
In [58]: fig = plt.figure(figsize = (18,3))

         colors = ["#bd5db0","#70a845","#727bcc","#b49242","#cc566c","#4aad92","#ca6037"]

         li_annees = inf_data.columns[4:]
         x=li_annees
         xlab=[my_str[:11]+"..." if len(my_str)>11 else my_str for my_str in x]
         y1=inf_data[li_annees].loc["null"]*100/data_c.shape[0] # % de valeurs nulles
         y2=inf_data[li_annees].loc["count"] # nombre de valeurs

         plot1 = plt.subplot(1,2,1)
         basic_plot("bar", plot1, xlab, y1, xlab[::4], "", "% NaN",  "% Données manquantes\n(Da

         plot2 = plt.subplot(1,2,2)
         basic_plot("bar", plot2, xlab, y2, xlab[::4], "", "Nbe non nul",  "Nbe d'entrées\n(Da

         plt.show()
```



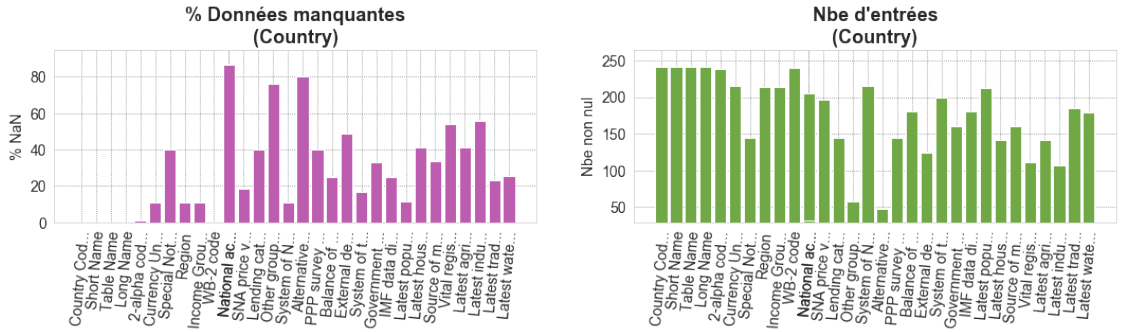
In [59]:
```
# nbe de remplissage minimum/maximum et année correspondante
sel_data = inf_data.loc["count"][4:]
val_min = sel_data.min()
val_max = sel_data.max()

print("- année nbe entrées min, nbe entrées min : \n{}, \n{:.0f}, soit {:.3f}%"\
      .format(sel_data.index[sel_data==val_min], \
              val_min, val_min*100/(data.shape[0])))

print("- année nbe entrées max, nbe entrées max : \n{}, \n{:.0f}, soit {:.3f}%"\
     .format(sel_data.index[sel_data==val_max], \
             val_max, val_max*100/(data.shape[0])))
```

```
- année nbe entrées min, nbe entrées min :
Index(['2017'], dtype='object'),
143, soit 0.016%
- année nbe entrées max, nbe entrées max :
Index(['2010'], dtype='object'),
242442, soit 27.335%
```

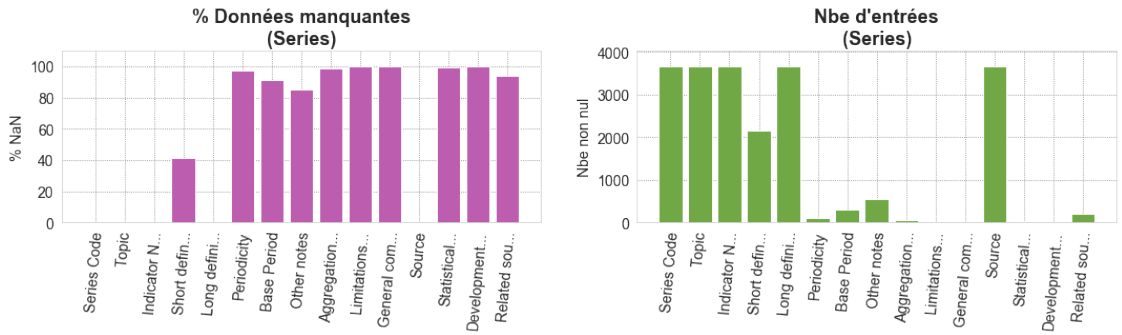
- La table "Data" donnant la valeur d'un indicateur pour une année comporte environ 86 % de données non renseignées.
- Environ 60% de l'ensemble des couples Indicateur/Pays n'a aucune valeur renseignée.
- L'année la mieux renseignée est l'année 2010 (27% des couples Indicateur/Pays, soit plus de 242 000 valeurs), et les moins renseignées sont les années 2016 et 2017 (respectivement 1,8% et 0,016% des couples, soit 16460 et 143 valeurs)

In [60]:
```
fig = plt.figure(figsize = (18,3))

x=inf_country.columns
xlab=[my_str[:11]+"..." if len(my_str)>11 else my_str for my_str in x]
y1=inf_country.loc["null"]*100/country.shape[0]  # % de valeurs nulles
y2=inf_country.loc["count"] # nombre de valeurs

plot1 = plt.subplot(1,2,1)
basic_plot("bar", plot1, xlab, y1, xlab, "", "% NaN",  "% Données manquantes\n(Country
```

```
plot2 = plt.subplot(1,2,2)
basic_plot("bar", plot2, xlab, y2, xlab, "", "Nbe non nul",  "Nbe d'entrées\n(Country)
plt.show()
```



In [61]:
```
fig = plt.figure(figsize = (18,3))

x=inf_series.columns
xlab=[my_str[:11]+"..." if len(my_str)>11 else my_str for my_str in x]
y1=inf_series.loc["null"]*100/series.shape[0]  # % de valeurs nulles
y2=inf_series.loc["count"] # nombre de valeurs

plot1 = plt.subplot(1,2,1)
basic_plot("bar", plot1, xlab, y1, xlab, "", "% NaN",  "% Données manquantes\n(Series)
plot2 = plt.subplot(1,2,2)
basic_plot("bar", plot2, xlab, y2, xlab, "", "Nbe non nul",  "Nbe d'entrées\n(Series)"
plt.show()
```

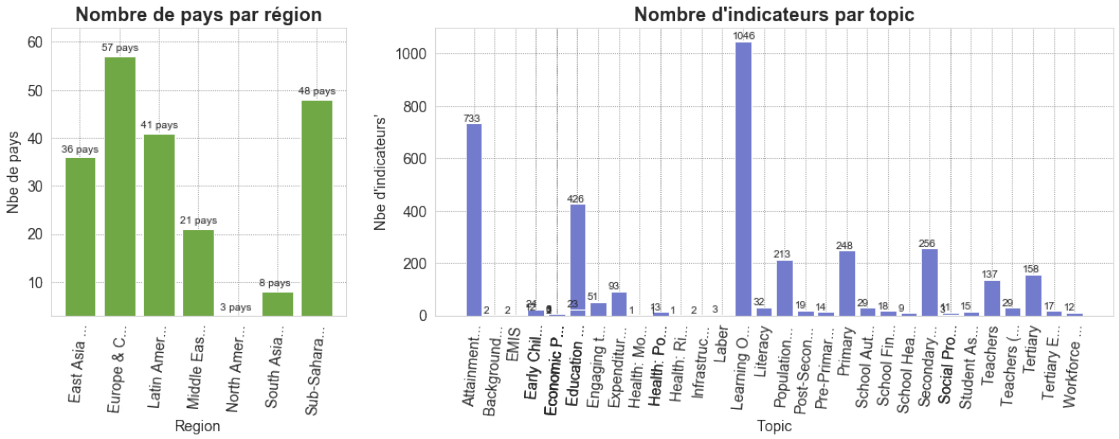


In [62]:
```
print("la table 'CountrySeries' contient {} entrées nulles".format(inf_cnt_ser.loc["nu
print("la table 'FootNote' contient {} entrées nulles".format(inf_footnote.loc["null"]
```

```
la table 'CountrySeries' contient 0 entrées nulles
la table 'FootNote' contient 0 entrées nulles
```

- Nombre de pays par région, et d'indicateurs par topic

```
In [63]: fig = plt.figure(figsize = (18,5))
         grid = plt.GridSpec(1, 3, wspace=0.3, hspace=0.3)

         df1 = country_c[['Country Name', 'Region']].groupby('Region').count()
         df2 = series_c[['Indicator Name', 'Topic']].groupby('Topic').count()
         li_region = list(df1.index)
         li_topic = list(df2.index)
         x1 = li_region
         x2 = li_topic
         xlab1 = [my_str[:10]+"..." if len(my_str)>8 else my_str for my_str in x1]
         xlab2 = [my_str[:10]+"..." if len(my_str)>8 else my_str for my_str in x2]
         y1 = df1.values.reshape(len(df1.values),)
         y2 = df2.values.reshape(len(df2.values),)

         ### Nombre de pays par région
         plot1 = plt.subplot(grid[0, 0])
         # basic_plot("bar", plot1, xlab1, y1, xlab1, "", "Région",  "Nombre de pays par régio
         plot1.bar(xlab1, y1, color = colors[1])
         plt.xticks(xlab1, rotation=85 , fontsize = 14), plt.yticks(fontsize = 14)
         plt.ylim(round(min(y1)*0.9), round(max(y1)*1.1))
         plot1.set_title("Nombre de pays par région", fontsize = 18, fontweight = 'bold')
         plot1.set_xlabel("Region", fontsize = 14), plot1.set_ylabel("Nbe de pays", fontsize =
         labels = [ '{:.0f} pays'.format(y1[i]) for i in range(len(y1))]
         for label,xlab1, y1 in zip(labels, xlab1, y1):
             plot1.annotate(label, xy=(xlab1, y1), xytext=(-17, 3),
                 textcoords='offset points', ha='left', va='bottom' )
         plt.grid(color='grey', linestyle='dotted')

         ### Nombre d'indicateurs par topic
         plot2 = plt.subplot(grid[0, 1:])
         # basic_plot("bar", plot2, xlab2, y2, xlab2, "", "Topic",  "Nombre d'indicateur par t
         plot2.bar(xlab2, y2, color = colors[2])
         plt.xticks(xlab2, rotation=85 , fontsize = 14), plt.yticks(fontsize = 14)
         plot2.set_title("Nombre d'indicateurs par topic", fontsize = 18, fontweight = 'bold')
         plot2.set_xlabel("Topic", fontsize = 14), plot2.set_ylabel("Nbe d'indicateurs'", fonts
         labels = [ '{:.0f}'.format(y2[i]) for i in range(len(y2))]
         for label,xlab2, y2 in zip(labels, xlab2, y2):
             plot2.annotate(label, xy=(xlab2, y2), xytext=(-10, 0),
                     rotation = 0, textcoords='offset points', ha='left', va='bottom' )
         plt.grid(color='grey', linestyle='dotted')

         plt.show()
```



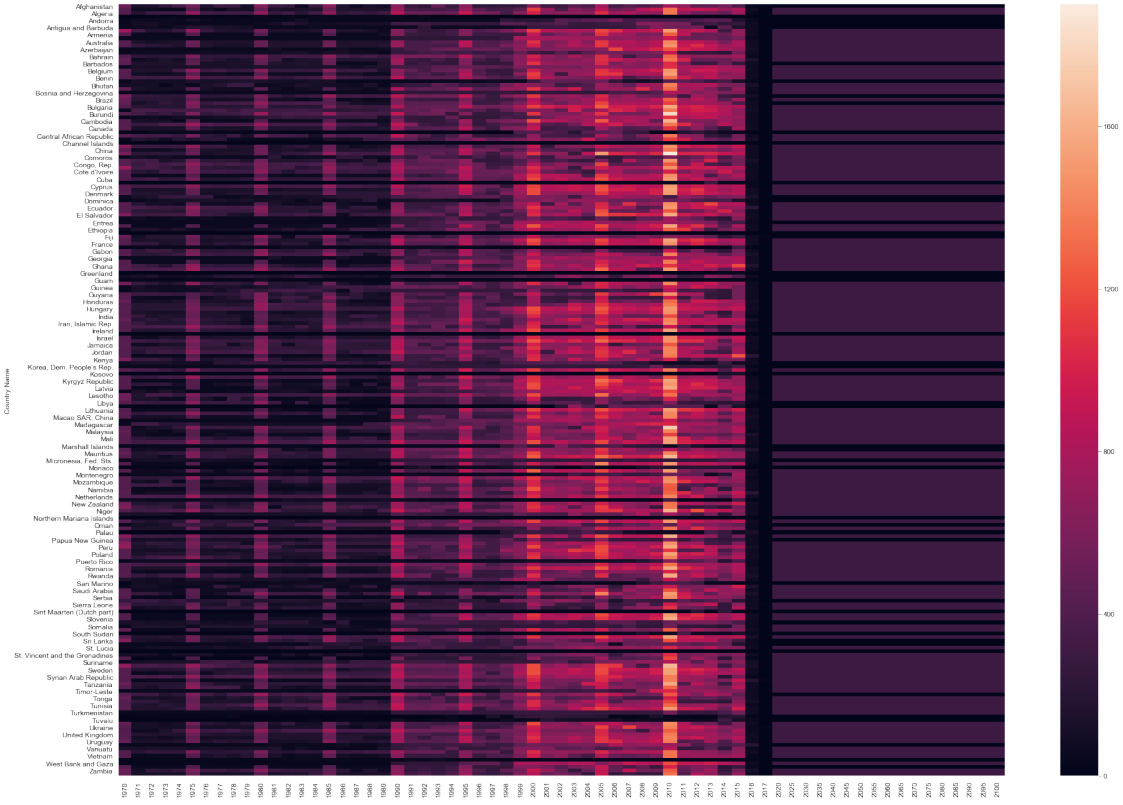
1.3 2. Exploration des données

Les indicateurs qui nous intéressent sont ceux des dernières années. On cherche à savoir : - combien d'indicateurs environ sont disponibles dans les dernières années - quels sont les pays qui ont le plus d'indicateurs disponibles dans les dernières années - quels sont les indicateurs le plus souvent disponible

1.3.1 Nombres d'indicateurs disponibles par pays et par année

Vision globale Pour avoir une idée de la proportion des indicateurs renseignés, on trace la carte de densité du nombre d'indicateur par pays et par année :

```
In [64]: ### Heatmap du nombre d'indicateurs non nuls (pays/années)

         nb_ind_cnt = data_c.groupby(['Country Name']).count()[li_annees]

         # Tableau des nombres d'indicateurs dispo pour chaque pays et chaque année
         fig = plt.figure(figsize = (28,20))
         heat_map = sns.heatmap(nb_ind_cnt)
```

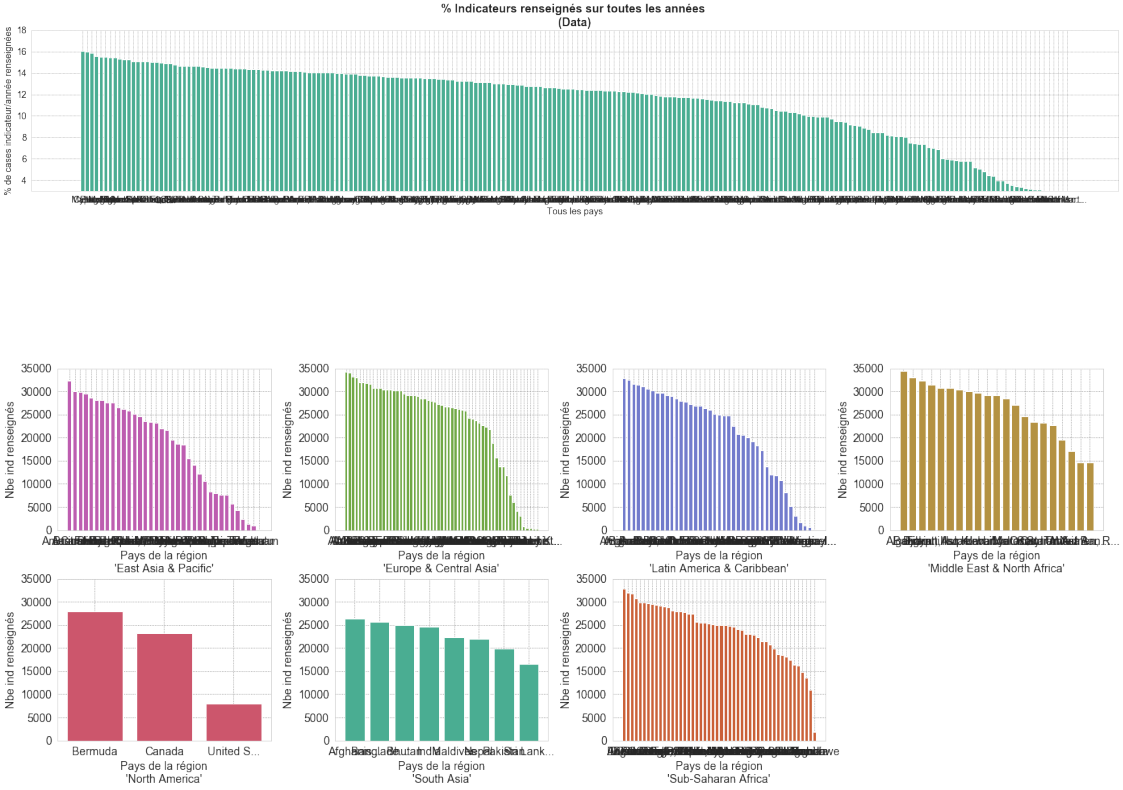


- Certains pays ont peu d'indicateurs, quelles que soient les années considérées (lignes sombres).
- Certaines années sont mieux renseignées (années multiples de 5)
- On distingue plusieurs plages de temps,
  - [1970,1989] : peu renseigné
  - [1990,1999] : assez bien renseigné
  - [2000,2015] : bien renseigné
  - [2016,2019] : quasiment pas renseigné

Exploration en vue de la suppression de colonnes (années) et lignes (pays) sous-remplis ou inutiles

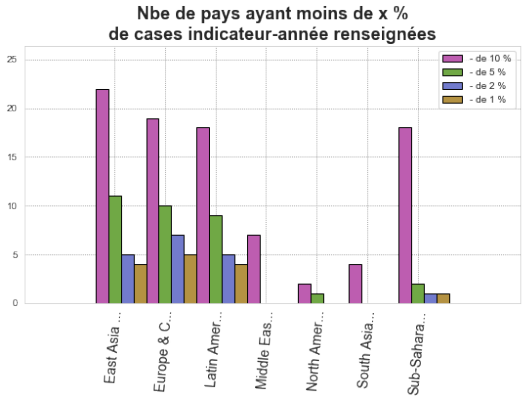
In [65]:
```
# tableau du pourcentage d'indicateurs renseignés par pays ()

fig1 = plt.figure(figsize = (30,10))
gp = data_c.groupby(['Country Name']).count()
nb_col_an = data_c[4:].shape[1]
nb_indic = series_c['Indicator Code'].shape[0]
nb_max = (nb_col_an*nb_indic)
tab = gp[gp.columns[3:]].sum(axis=1)*100/nb_max
tab.sort_values(ascending=False,inplace=True)
```

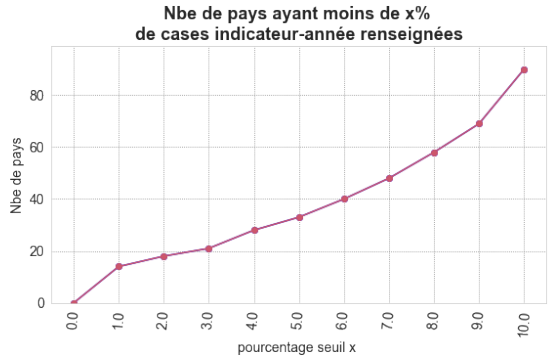
```
x=tab.index
xlab=[my_str[:8]+"..." if len(my_str)>8 else my_str for my_str in x]
y=tab
plot1 = plt.subplot(2,1,1)
basic_plot("bar", plot1, xlab, y, '', "Tous les pays", "% de cases indicateur/année re
plt.show()

# tableau du pourcentage d'indicateurs renseignés par pays et par région ()

fig2 = plt.figure(figsize = (30,10))
tab_df = [data_c.groupby(['Region', 'Country Name']).count().loc[reg][li_annees] for r
n = 4 # nombre de colonnes d'affichage en largeur
tab_plot = []
for i in range(len(tab_df)):
    x=tab_df[i].index
    xlab=[my_str[:8]+"..." if len(my_str)>8 else my_str for my_str in x]
    y=tab_df[i].sum(axis=1) # nombre d'indicateurs renseignés
    y.sort_values(ascending=False,inplace=True)
    tab_plot.append(plt.subplot((len(tab_df)+1)//n+1,n,i+1))
    basic_plot("bar", tab_plot[i], xlab, y, '', "Pays de la région\n'"+ li_region[i] +
    plt.ylim(0,35000)
plt.gcf().subplots_adjust(left = 0.1, bottom = 0.2, right = 0.7, top = 1.2, wspace = 0
plt.show()
```



Quelle que soit la région considérée, la plupart des pays ont plus de 5000 indicateurs renseignés, toutes années confondues.

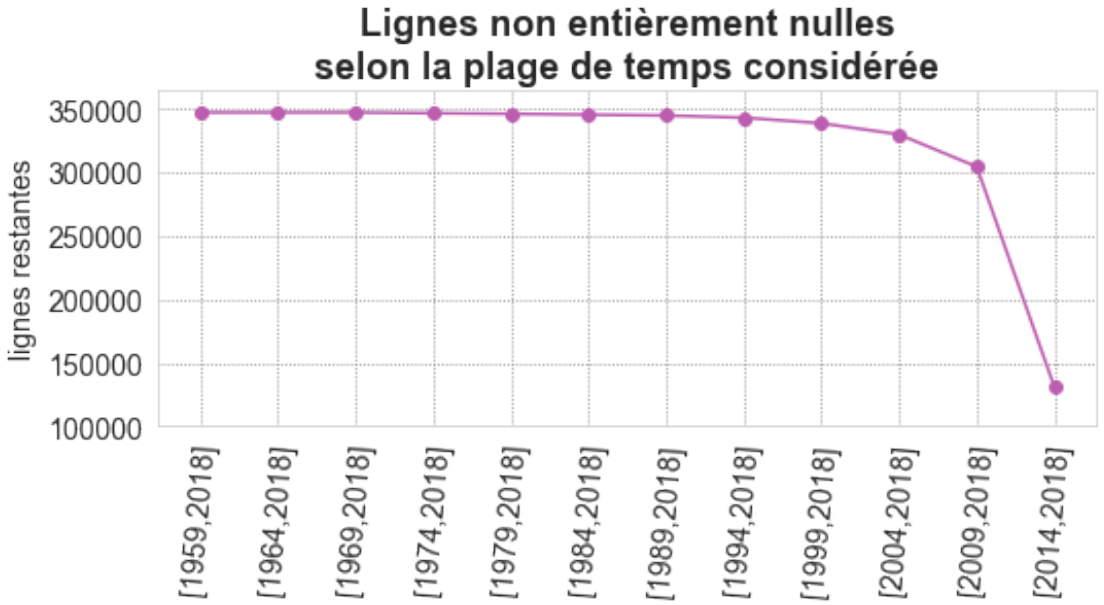
```
In [66]:  # Pays ayant moins de pct*100 % d'indicateurs renseignés, par région

          def nbe_pays_inf_pct (pct, nb_ind) :
              return [tab_df[i][tab_df[i].sum(axis=1)<pct*(nb_ind*len(li_annees))].shape[0] for

          fig = plt.figure(figsize = (20,5))

          plot1 = plt.subplot(1,2,1)
          barWidth = 0.25
          x = li_region
          xlab = [my_str[:10]+"..." if len(my_str)>8 else my_str for my_str in x]
          nb_indic = len(data_c["Indicator Code"].unique())
          i=0
          for pct in [0.1,0.05,0.02,0.01] :
              plot1.bar(np.arange(len(xlab))+i*barWidth, nbe_pays_inf_pct(pct,nb_indic),\
                        color=colors[i], width=barWidth, ec='k', label = "- de %.0f"%(pct*100) +
              i+=1
          plt.xticks([r + barWidth for r in range(len(xlab))], xlab, rotation=85 , fontsize = 14
          #plot1.set_xlabel(fontsize = 14), plot1.set_ylabel(fontsize = 14)
          plot1.set_title("Nbe de pays ayant moins de x %\nde cases indicateur-année renseignées
          plt.grid(color='grey', linestyle='dotted'), plt.legend()
          plt.margins(0.2), plt.subplots_adjust(bottom=0.15)

          plot2 = plt.subplot(1,2,2)
          x = np.linspace(0,10, 11)
          y = [sum(nbe_pays_inf_pct(val/100, nb_indic)) for val in x]
          plot2.plot(x,y, '-o', color = 'b')
          basic_plot("plot", plot2, x, y, x, "pourcentage seuil x ",\
                     "Nbe de pays",  "Nbe de pays ayant moins de x% \nde cases indicateur-année
          plt.show()

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: FutureWarning: elementwise
```





On pourrait enlever les pays qui ont moins de 2% de leurs indicateurs renseignés (moins de 20 pays).

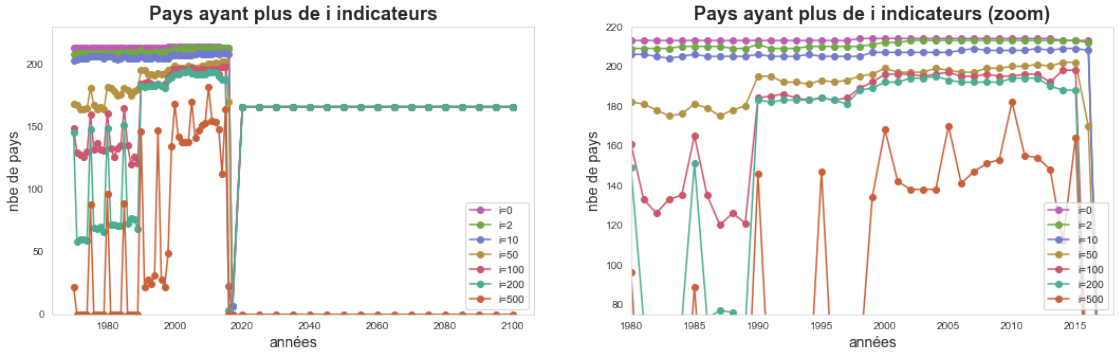
```
In [67]: # Nombre de lignes restantes (non entièrement vides) après sélection des années

         borne_annee_min = [1959,1964,1969,1974,1979,1984,1989,1994,1999,2004,2009,2014]
         y=[]
         for annee_min in borne_annee_min:
             new_li_annees = [annee for annee in list(li_annees) if int(annee)>annee_min and in
             y.append(data_c[new_li_annees].dropna(how = 'all', inplace = False).shape[0])

         fig = plt.figure(figsize = (18,3))

         x=borne_annee_min
         xlab=["["+str(annee)+",2018]" for annee in x]

         plot1 = plt.subplot(1,2,1)
         basic_plot("plot", plot1, xlab, y, xlab, "", "lignes restantes", \
                    "Lignes non entièrement nulles\nselon la plage de temps considérée", 0)
         plot1.set_ylim(100000,data_c[li_annees].dropna(how = 'all', inplace = False).shape[0]+

Out[67]: (100000, 364715.4)
```



Lorsque l'on écarte les données avant 1990 environ, le nombre de lignes entièrement nulles commence à croître. Or la part de la population ayant entre 25 et 26 ans aujourd'hui est née entre 1993 et 1994. Pour notre étude, il n'est pas pertinent d'utiliser des données avant ces années. On retire donc toutes les années avant 1990. On retirera ensuite les lignes qui ne contiennent pas de données pour les années 1990-2018.

In [68]:
```
# Nbe de pays ayant un nbe d'indicateurs supérieur à n en fonction des années

nb_ind_cnt = data_c.groupby(['Country Name']).count()[li_annees]

def calc_nb_pay_rens (tab_n, years): # prend un tableau de seuils entiers et un table
    tab = [ [nb_ind_cnt[nb_ind_cnt[str(i)]>j].index.size for i in years] for j in tab_
    res = np.array(tab).T
    return res

fig = plt.figure(figsize = (18,5))

countrys = [nb_ind_cnt.index[i][0] + " - " + nb_ind_cnt.index[i][1] for i in range(nb_
tab_n = [0,2,10,50,100, 200, 500]
x =  [int(y) for y in li_annees]
y = calc_nb_pay_rens(tab_n,x)
colors = ["#bd5db0","#70a845","#727bcc","#b49242","#cc566c","#4aad92","#ca6037"]
labels = ["i="+str(i) for i in tab_n]

plot1 = plt.subplot(1,2,1)
[plot1.plot(x, y[:,i], '-o', label = labels[i], color = colors[i]) for i in range(len
plot1.set_ylim(0,230)
plot1.set_xlabel("années", fontsize = 14), plot1.set_ylabel("nbe de pays", fontsize =
```

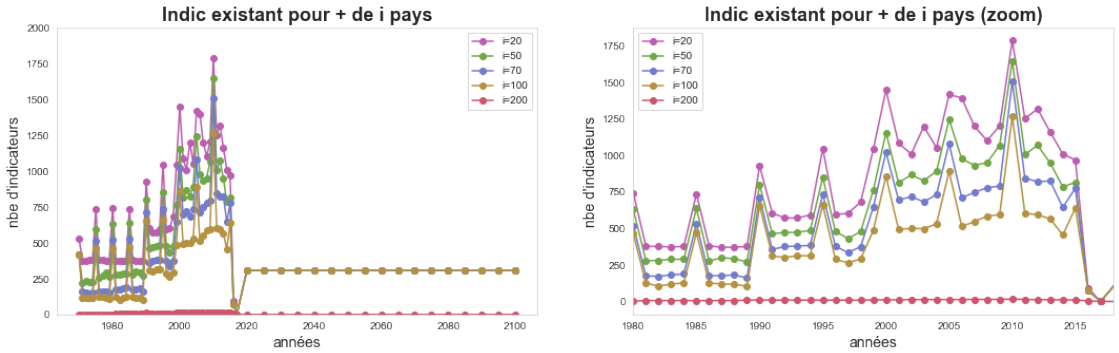
```
plot1.set_title("Pays ayant plus de i indicateurs", fontsize = 18, fontweight = 'bold
plot1.legend(loc = 'lower right'), plt.grid()

plot2 = plt.subplot(1,2,2)
[plot2.plot(x, y[:,i], '-o', label = labels[i], color = colors[i]) for i in range(len
plot2.set_xlim(1980,2018), plot2.set_ylim(75,220)
plot2.set_xlabel("années", fontsize = 14), plot2.set_ylabel("nbe de pays", fontsize =
plot2.set_title("Pays ayant plus de i indicateurs (zoom)", fontsize = 18, fontweight =
plot2.legend(loc = 'lower right'), plt.grid()
plt.show()
```



- Plus de la moitié des pays ont plus de 500 indicateurs remplis pour chaque année à partir de 2000.
- Plus de 80% des pays ont plus de 200 indicateurs remplis chaque année à partir de 2000.
- Tous les pays ont au moins 1 indicateur
- 167 pays ont des projections d'indicateurs (entre 200 et 500)
- les données sont très rares pour l'année 2017 (7 indicateurs au plus)

In [69]:
```
# Nbe d'indicateurs existant pour plus de n pays selon les années

nb_pay_cnt = data_c.groupby(['Indicator Name']).count()[li_annees]

def calc_nb_ind_rens (tab_n, years): # prend un tableau de seuils entiers et un table
    tab = [ [nb_pay_cnt[nb_pay_cnt[str(i)]>j].index.size for i in years] for j in tab
    res = np.array(tab).T
    return res

fig = plt.figure(figsize = (18,5))

countrys = [nb_ind_cnt.index[i][0] + " - " + nb_ind_cnt.index[i][1] for i in range(nb
tab_n = [20,50,70,100, 200]
x =  [int(y) for y in li_annees] # liste des anneés
y = calc_nb_ind_rens(tab_n,x)
colors = ["#bd5db0","#70a845","#727bcc","#b49242","#cc566c","#4aad92","#ca6037"]
```

```
labels = ["i="+str(i) for i in tab_n]

plot1 = plt.subplot(1,2,1)
[plot1.plot(x, y[:,i], '-o', label = labels[i], color = colors[i]) for i in range(len
plot1.set_ylim(0,2000)
plot1.set_xlabel("années", fontsize = 14), plot1.set_ylabel("nbe d'indicateurs", fonts
plot1.set_title("Indic existant pour + de i pays", fontsize = 18, fontweight = 'bold'
plot1.legend(loc = 'upper right'), plt.grid()

plot2 = plt.subplot(1,2,2)
[plot2.plot(x, y[:,i], '-o', label = labels[i], color = colors[i]) for i in range(len
plot2.set_xlim(1980,2018), #plot2.set_ylim(75,220)
plot2.set_xlabel("années", fontsize = 14), plot2.set_ylabel("nbe d'indicateurs", fonts
plot2.set_title("Indic existant pour + de i pays (zoom)", fontsize = 18, fontweight =
plot2.legend(loc = 'upper left'), plt.grid()
plt.show()
```



- Environ 500 indicateurs sont renseignés chaque année pour plus de 100 pays (la moitié des pays) de 2000 à 2015.
- Seuls 20 pays environ ont plus de 1000 indicateurs renseignés chaque année.
- 167 pays sur les 214 (voir graphes précédents) ont des projections pour 308 indicateurs sur les 3665.

Suppression de lignes et de colonnes L'exploration ci-dessus nous permet de prendre des décisions par rapport aux années et aux pays sous-remplis : * On garde les années de 1990 à 2015 inclus, puis on élimine les lignes entièrement nulles (sans indicateur renseigné). * On garde seulement les pays qui ont plus de 2 % des cases indicateur/année remplies

```
In [70]: # Suppression des colonnes années inutiles, et des lignes entièrement vides après opé
         li_col_supp = [annee for annee in list(li_annees) if int(annee)<1990 or int(annee)>201
         print("Taille de la dataframe 'data_c' avant suppression des colonnes : ", data_c.shap
         data_c.drop(columns = li_col_supp, inplace = True)
         print("Taille de la dataframe 'data_c' après suppression des colonnes : ", data_c.shap
         li_annees_sel = sorted(Diff(li_annees, li_col_supp)[0]) # colonnes à conserver
```

```
        lign_supp = sorted(Diff(data_c[li_annees_sel].dropna\
                                (how='all', axis = 0, inplace=False).index, data_c.index)[1]) # i
        data_c.drop(index = lign_supp, inplace = True)
        print("Taille de la dataframe 'data_c' après suppression des lignes vides : ", data_c
```

```
Taille de la dataframe 'data_c' avant suppression des colonnes :  (784310, 71)
Taille de la dataframe 'data_c' après suppression des colonnes :  (784310, 32)
Taille de la dataframe 'data_c' après suppression des lignes vides :  (344238, 32)
```

```
In [71]: # Suppression des pays ayant moins de 2% de cases indicateur/année renseignées
         # (soit environ 2300 pour 32 années et 3665 indicateurs)
         gp = data_c.groupby(['Country Name']).count() # nbe de cases renseignées pour chaque
         gp_sum = gp[gp.columns[5:]].sum(axis=1) # nbe de cases renseignées pour chaque pays (
         pays_supp = list(gp_sum[gp_sum<2300].index) # liste des pays à supprimer (16)
         tab_index_supp = [list(data_c[data_c["Country Name"]==pays].index) for pays in pays_su
         li_lign_supp = sorted([j for index in tab_index_supp for j in index]) # liste des lig
         print("Suppression de {} pays, (soit {} lignes) à savoir :\n{}".format(len(pays_supp)
         print("Taille de la dataframe 'data_c' avant suppression : ", data_c.shape)
         data_c.drop(index = li_lign_supp, inplace = True)
         print("Taille de la dataframe 'data_c' après suppression : ", data_c.shape)
```

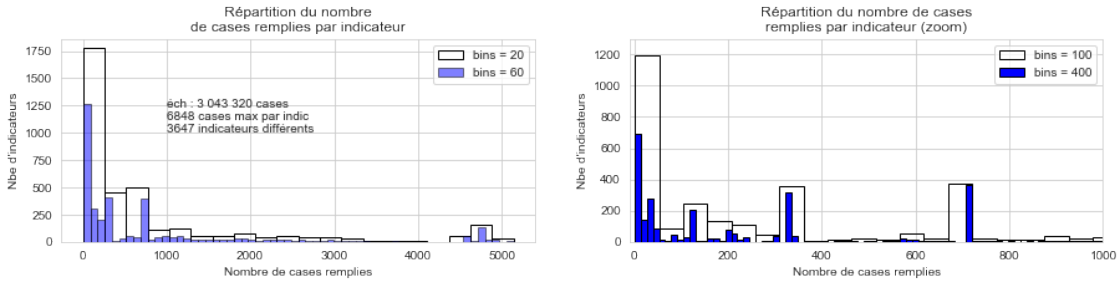
```
Suppression de 16 pays, (soit 2805 lignes) à savoir :
['American Samoa', 'Channel Islands', 'Curacao', 'Faroe Islands', 'French Polynesia', 'Greenla
Taille de la dataframe 'data_c' avant suppression :  (344238, 32)
Taille de la dataframe 'data_c' après suppression :  (341433, 32)
```

Exploration en vue de la suppression de lignes (indicateurspays) sous-remplis On s'intéresse maintenant aux indicateurs peu renseignés. Y a-t-il des indicateurs trop peu renseignés dans la plage de temps qui nous intéresse (1990-2015) ? Le manque d'information concerne-t-il en particulier les années ou les pays ?

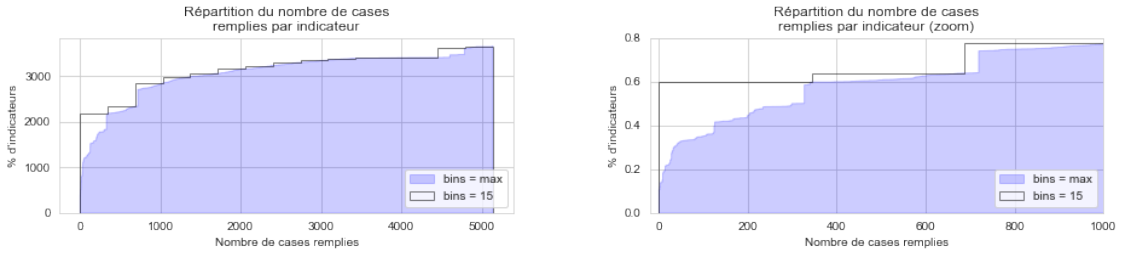
```
In [72]: # Exploration des indicateurs disponibles pour un nombre insuffisant de pays ou d'ann
         gp = data_c.groupby(['Indicator Code']).count() # nbe de cases renseignées pour chaqu
         gp_sum = gp[gp.columns[5:]].sum(axis=1) # nbe de cases renseignées pour chaque indica

         #np.sum(gp_sum) # 3043320 cases en tout
         #len(data_c['Indicator Code'].unique()) # 3647 indicateurs différents
```

```
In [73]: #---------------------------------
         fig1 = plt.figure(figsize = (15,3))
         plot1 = plt.subplot(1,2,1)
         # échantillon de 3043320 cases remplies, 6848 cases maximum par indicateur
         plot1.hist(gp_sum, bins = 20, fc = 'None', ec = 'k', label='bins = 20')
         plot1.hist(gp_sum, bins = 60, fc = 'b', alpha = 0.5, ec = 'k', label='bins = 60')
         plt.xlabel("Nombre de cases remplies"), plt.ylabel("Nbe d'indicateurs")
         plt.title('Répartition du nombre\nde cases remplies par indicateur')
```

```
plt.grid(True), plt.legend()
plt.text(1000, 1000,"éch : 3 043 320 cases\n6848 cases max par indic\n3647 indicateurs
#plt.xlim(40, 160), plt.ylim(0, 0.03)
#---
plot2 = plt.subplot(1,2,2)
plot2.hist(gp_sum, bins = 100, fc = 'None', ec = 'k', label='bins = 100')
plot2.hist(gp_sum, bins = 400, fc = 'b', ec = 'k',  label='bins = 400')
plt.xlabel("Nombre de cases remplies"), plt.ylabel("Nbe d'indicateurs")
plt.xlim(-10, 1000), plt.ylim(0, 1300), plt.grid(True) , plt.legend()
plt.title('Répartition du nombre de cases\nremplies par indicateur (zoom)')
plt.show()
#---------------------------------
```

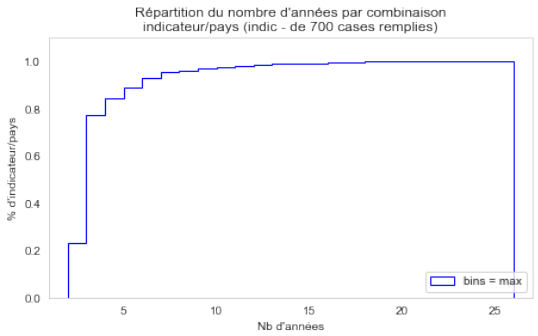


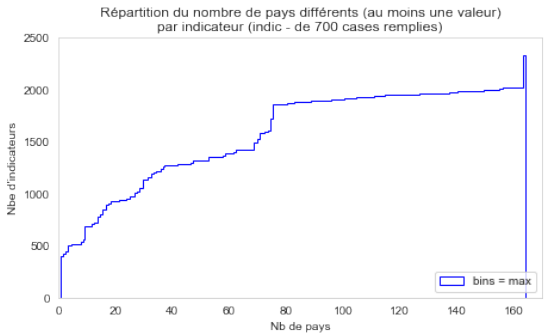
In [74]:
```
#---------------------------------
fig2 = plt.figure(figsize = (20,2))
plot3 = plt.subplot(1,2,1)
plot3.hist(gp_sum, bins = 3700, fc = 'b', alpha = 0.2, ec = 'b', density=False,\
           histtype='stepfilled', cumulative=True, label='bins = max')
plot3.hist(gp_sum, bins = 15, fc = 'r', alpha = 0.5, ec = 'k', density=False,\
           histtype='step', cumulative=True, label='bins = 15')
plt.xlabel("Nombre de cases remplies"), plt.ylabel("% d'indicateurs")
plt.grid(True), plt.legend(loc='lower right')#, plt.xlim(-20, 1000), plt.ylim(0, 1000
plt.title('Répartition du nombre de cases\nremplies par indicateur')
#---
plot4 = plt.subplot(1,2,2)
plot4.hist(gp_sum, bins = 3700, fc = 'b', alpha = 0.2, ec = 'b', density=True,\
           histtype='stepfilled', cumulative=True, label='bins = max')
plot4.hist(gp_sum, bins = 15, fc = 'r', alpha = 0.5, ec = 'k', density=True,\
           histtype='step', cumulative=True, label='bins = 15')
plt.xlabel("Nombre de cases remplies"), plt.ylabel("% d'indicateurs")
plt.grid(True), plt.legend(loc='lower right'), plt.xlim(-20, 1000), plt.ylim(0, 0.8)
plt.title('Répartition du nombre de cases\nremplies par indicateur (zoom)')

plt.gcf().subplots_adjust(left = 0.1, bottom = 0.2, right = 0.7, top = 1.2, wspace =
plt.show()
#---------------------------------
```



Il y a plus de 2300 indicateurs sur 3647 (63%) qui ont moins de 700 cases remplies. Intéressons-nous au taux de remplissage de ces indicateurs en particulier.

```
In [75]: li_ind_m700 = gp_sum[gp_sum.values<700].index # liste des codes des 2332 indicateurs
         df_ind_m700 = data_c[data_c["Indicator Code"].isin(li_ind_m700)] # 126805 lignes conc
         ind_uni = df_ind_m700["Indicator Code"].unique() # liste des noms des indicateurs con
         # nombre d'années calc par indic et par pays
         nb_ann = df_ind_m700[df_ind_m700.columns[5:]].count(axis=1)  # (25 bins de 2 à 26)
         # nombre de pays différents par indicateur (pour au moins une année)
         tab = df_ind_m700.groupby(["Indicator Code", "Country Code"])[li_annees_sel].count().s
         nb_pay = [len(tab.loc[code]) for code in ind_uni] # liste des nbes de pays ayant au -
         #nb_pay =  [len(tab.loc[code][(tab.loc[code]!=0).values]) for code in ind_uni]  # en

         #----------------
         fig = plt.figure(figsize = (16,4))
         #----------------
         plot1 = plt.subplot(1,2,1)
         plot1.hist(nb_ann, bins = 24, fc = 'b', alpha = 1, ec = 'b', density=True,\
                    histtype='step', cumulative=True, label='bins = max')
         # plot1.hist(nb_ann, bins = 15, fc = 'r', alpha = 0.5, ec = 'k', density=True,\
         #            histtype='step', cumulative=True, label='bins = 15')
         plt.title("Répartition du nombre d'années par combinaison\nindicateur/pays (indic - de
         plt.xlabel("Nb d'années"), plt.ylabel("% d'indicateur/pays")
         plt.xlim(1, 27), plt.ylim(0, 1.1), plt.grid(), plt.legend(loc='lower right')

         #----------------
         plot2 = plt.subplot(1,2,2)
         plot2.hist(nb_pay, bins = 215, fc = 'b', alpha = 1, ec = 'b', density=False,\
                    histtype='step', cumulative=True, label='bins = max')
         # plot1.hist(nb_ann, bins = 15, fc = 'r', alpha = 0.5, ec = 'k', density=True,\
         #            histtype='step', cumulative=True, label='bins = 15')
         plt.title('Répartition du nombre de pays différents (au moins une valeur)\npar indicat
         plt.xlabel("Nb de pays"), plt.ylabel("Nbe d'indicateurs")
         plt.xlim(0, 170), plt.ylim(0, 2500), plt.grid(), plt.legend(loc='lower right')

         plt.show()
```





Précisions au sujet des indicateurs peu remplis (moins de 700 cases sur 26*214 = 5564 cases) - combien d'années sont calculées par indicateur ? -> La grande majorité des combinaisons pays/indicateur sont calculés pour moins de 4 années. 20% des indicateurs nationaux sont calculés pour seulement 2 années entre 1990 et 2015. On choisit de les éliminer. - combien de pays différents par indicateur ? -> Il n'y a pas de ligne de démarcation nette qui nous permettrait de choisir les indicateurs. Sur les 2300 indicateurs restant, la moitié ne sont disponibles que pour moins de 30 pays. On choisit de ne pas éliminer d'indicateurs sur ce critère.

Suppression des lignes indicateur/pays calculés pour seulement 1 ou 2 années dans la plage [1990,2015]

```
In [76]: ######### ATTENTION, éliminé toutes les combis ayant moins de 3 années calculées
         li_supp = data_c[data_c[data_c.columns[6:]].count(axis=1)<3].index # 37634, 76279 = 1
         print("Taille de la dataframe 'data_c' avant suppression des lignes indicateurs : ", c
         data_c.drop(index = li_supp, inplace = True)
         print("Taille de la dataframe 'data_c' après suppression des lignes : ", data_c.shape)
```

```
Taille de la dataframe 'data_c' avant suppression des lignes indicateurs :  (341433, 32)
Taille de la dataframe 'data_c' après suppression des lignes :  (227520, 32)
```

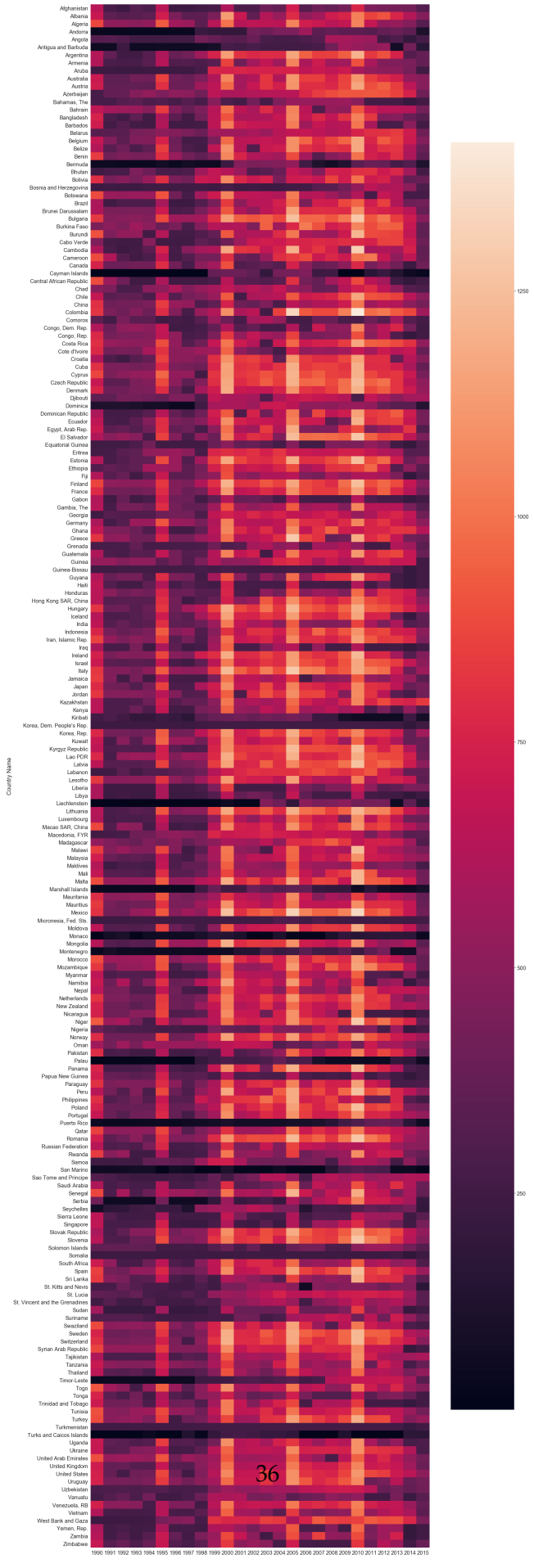
```
In [77]: ### Nouvelle heatmap du nombre d'indicateurs non nuls (pays/années)

         nb_ind_cnt = data_c.groupby(['Country Name']).count()[li_annees_sel]
         nb_ind_cnt.head()
```

Out[77]:

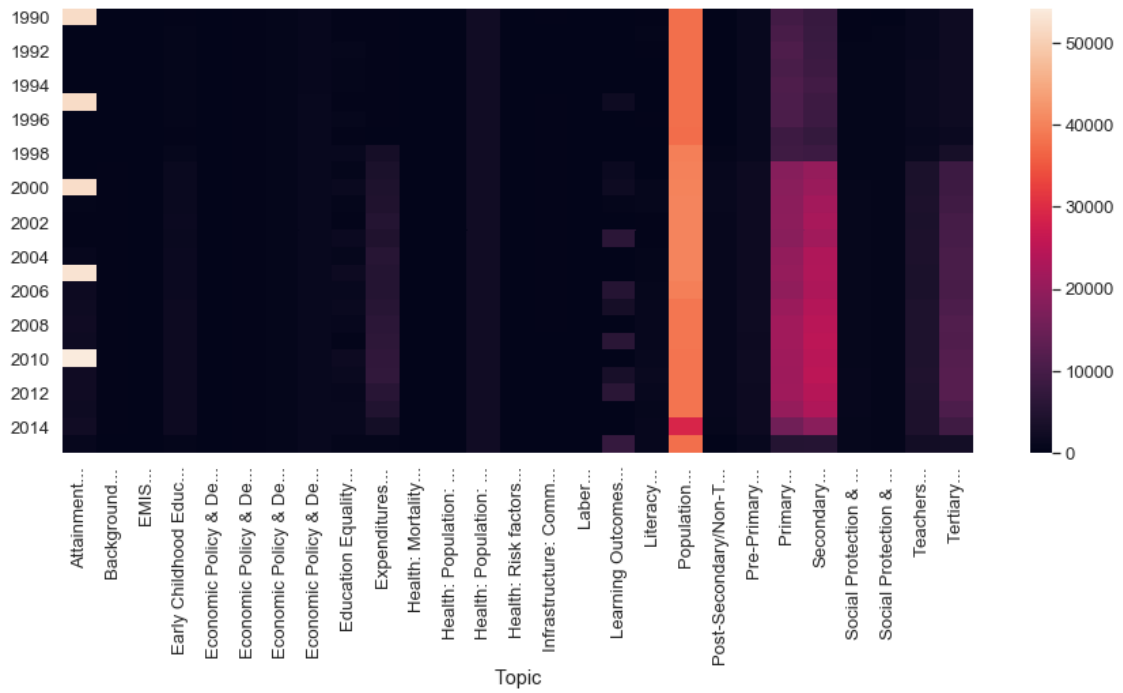
Country Name	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001
Afghanistan	648	277	236	320	285	646	236	236	247	266	612	294
Albania	698	351	328	331	352	725	451	340	347	454	1130	704
Algeria	875	518	520	497	524	930	575	552	348	645	1002	632
Andorra	43	42	42	42	42	42	43	44	225	200	200	199
Angola	315	324	299	249	248	248	249	250	423	381	333	327

```
In [78]: data_c.to_csv('../DONNEES/data_c.csv', index = False)
```

In [79]:
```
# Tableau des nombres d'indicateurs dispo pour chaque pays et chaque année
sns.set(font_scale=1.8)
fig = plt.figure(figsize = (28,105))
heat_map = sns.heatmap(nb_ind_cnt)
```


Country Name
Afghanistan
Albania
Algeria
Andorra
Angola
Antigua and Barbuda
Argentina
Armenia
Aruba
Australia
Austria
Azerbaijan
Bahamas, The
Bahrain
Bangladesh
Barbados
Belarus
Belgium
Belize
Benin
Bermuda
Bhutan
Bolivia
Bosnia and Herzegovina
Botswana
Brazil
Brunei Darussalam
Bulgaria
Burkina Faso
Burundi
Cabo Verde
Cambodia
Cameroon
Canada
Cayman Islands
Central African Republic
Chad
Chile
China
Colombia
Comoros
Congo, Dem. Rep.
Congo, Rep.
Costa Rica
Cote d'Ivoire
Croatia
Cuba
Cyprus
Czech Republic
Denmark
Djibouti
Dominica
Dominican Republic
Ecuador
Egypt, Arab Rep.
El Salvador
Equatorial Guinea
Eritrea
Estonia
Ethiopia
Fiji
Finland
France
Gabon
Gambia, The
Georgia
Germany
Ghana
Greece
Grenada
Guatemala
Guinea
Guinea-Bissau
Guyana
Haiti
Honduras
Hong Kong SAR, China
Hungary
Iceland
India
Indonesia
Iran, Islamic Rep.
Iraq
Ireland
Israel
Italy
Jamaica
Japan
Jordan
Kazakhstan
Kenya
Kiribati
Korea, Dem. People's Rep.
Korea, Rep.
Kuwait
Kyrgyz Republic
Lao PDR
Latvia
Lebanon
Lesotho
Liberia
Libya
Liechtenstein
Lithuania
Luxembourg
Macao SAR, China
Macedonia, FYR
Madagascar
Malawi
Malaysia
Maldives
Mali
Malta
Marshall Islands
Mauritania
Mauritius
Mexico
Micronesia, Fed. Sts.
Moldova
Monaco
Mongolia
Montenegro
Morocco
Mozambique
Myanmar
Namibia
Nepal
Netherlands
New Zealand
Nicaragua
Niger
Nigeria
Norway
Oman
Pakistan
Palau
Panama
Papua New Guinea
Paraguay
Peru
Philippines
Poland
Portugal
Puerto Rico
Qatar
Romania
Russian Federation
Rwanda
Samoa
San Marino
Sao Tome and Principe
Saudi Arabia
Senegal
Serbia
Seychelles
Sierra Leone
Singapore
Slovak Republic
Slovenia
Solomon Islands
Somalia
South Africa
Spain
Sri Lanka
St. Kitts and Nevis
St. Lucia
St. Vincent and the Grenadines
Sudan
Suriname
Swaziland
Sweden
Switzerland
Syrian Arab Republic
Tajikistan
Tanzania
Thailand
Timor-Leste
Togo
Tonga
Trinidad and Tobago
Tunisia
Turkey
Turkmenistan
Turks and Caicos Islands
Uganda
Ukraine
United Arab Emirates
United Kingdom
United States
Uruguay
Uzbekistan
Vanuatu
Venezuela, RB
Vietnam
West Bank and Gaza
Yemen, Rep.
Zambia
Zimbabwe
1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015
1250
1000
750
500
250

```
In [80]: # Enquête sur les 10 topics disparus suite à l'élimination des pays, années, indicate
         li_top_data_c = data_c["Topic"].unique()
         li_top_data = pd.merge(data, series, left_on="Indicator Code", right_on="Series Code"
         li_topics_supp = list(Diff(li_top_data, li_top_data_c)[0])
         data_plus = pd.merge(data, series, left_on="Indicator Code", right_on="Series Code")

In [81]: li_topics_supp

Out[81]: ['Early Child Development (SABER)',
          'Student Assessment (SABER)',
          'Teachers (SABER)',
          'Education Management Information Systems (SABER)',
          'School Finance (SABER)',
          'School Health and School Feeding (SABER)',
          'School Autonomy and Accountability (SABER)',
          'Engaging the Private Sector (SABER)',
          'Workforce Development (SABER)',
          'Tertiary Education (SABER)']

In [82]: # sélection des index des topics éliminés
         selec_ind = data_plus.isin({'Topic': li_topics_supp}).replace(False, np.nan).dropna(ho

In [83]: # Sur quelles années portent les données des topics éliminés ?
         #data_plus.iloc[selec_ind][li_annees].count()
         # 2009 105 - 2010  176- 2011 116 - 2012 701 - 2013  785 - 2014  502- 2015   410 - 201
         # seulement des données de 2009 à 2017

In [84]: # Tableau des nombres de valeurs d'indicateurs dispo par topic
         sns.set(font_scale=1.2)
         gb = data_c[['Topic'] + li_annees_sel].groupby('Topic').count()#.sum(axis = 1)  # 27
         fig = plt.figure(figsize = (15,6))
         heat_map = sns.heatmap(gb.T)
         # format text labels
         xticklabels = [] # fmt = '{:0.2f}'
         for item in heat_map.get_xticklabels():
             item.set_text(item.get_text()[:20]+"...") #(fmt.format(float(item.get_text())))
             xticklabels += [item]
         heat_map.set_xticklabels(xticklabels)
         plt.show()
```



Multiindexage de la dataframe data_c et permutation des axes Afin de faciliter la recherche des outliers, les calculs de corrélation entre les différents indicateurs, l'ANOVA et les tracés des indicateurs pertinents, on effectue les opérations suivantes : - multiindexage de lignes par 'Topic', 'Indicator Code' - indexation des années en lignes plutôt qu'en colonnes grâce à un nouvel index 'Year' - multiindexage de 'Region' et 'Country Code' en colonnes

```
In [85]: # 'Topic', 'Indicator Code','Indicator Name', 'Region', 'Country Name', 'Country Code
         data_c_mod = data_c[['Topic','Indicator Code','Region','Country Name']+li_annees_sel]
         data_c_mod.set_index(['Topic','Indicator Code','Region','Country Name'], inplace = Tru
         data_c_mod.columns = pd.MultiIndex.from_product([data_c_mod.columns, ['val']], names =
         data_c_mod.columns = data_c_mod.columns.swaplevel(0, 1)
         data_c_mod = data_c_mod.unstack(['Region','Country Name'])
         data_c_mod.columns = data_c_mod.columns.droplevel()
         data_c_mod.sort_index(inplace=True)
         data_c_mod = data_c_mod.dropna(how = 'all', inplace = False, axis = 0)
         data_c_mod = data_c_mod.T.dropna(how = 'all', inplace = False, axis = 0)
         data_c_mod.sort_index(inplace=True)
         data_c_mod.shape # 5148,2310 -> 11ã891ã880 cases
         data_c_mod.head()
```

```
Out[85]: Topic                                                Attainment
         Indicator Code                          BAR.NOED.1519.FE.ZS BAR.NOED.1519.ZS BAR.NO
         Year Region              Country Name
         1990 East Asia & Pacific Australia                       0.1                 0.1
```

	Brunei Darussalam	20.1	19.7	
	Cambodia	38.7	32.8	
	China	4.5	5.3	
	Fiji	1.2	1.1	

Topic				
Indicator Code		BAR.POP.7074.FE	BAR.POP.75UP	BAR.POP.75UP.
Year	Region	Country Name		
1990	East Asia & Pacific	Australia	267.0	744.0	467
		Brunei Darussalam	1.0	2.0	1
		Cambodia	48.0	65.0	38
		China	9895.0	19008.0	11428
		Fiji	3.0	6.0	3

Topic			
Indicator Code		BAR.PRM.ICMP.3034.FE.ZS	BAR.PRM.ICMP.3034.Z
Year	Region	Country Name	
1990	East Asia & Pacific	Australia	3.5	3
		Brunei Darussalam	20.0	21
		Cambodia	53.6	58
		China	42.0	28
		Fiji	28.8	29

Topic				
Indicator Code		BAR.PRM.SCHL.6569.FE	BAR.PRM.SCHL.7074	BAR
Year	Region	Country Name		
1990	East Asia & Pacific	Australia	5.7	5.8	
		Brunei Darussalam	0.5	1.4	
		Cambodia	0.3	1.0	
		China	0.7	1.0	
		Fiji	3.9	3.8	

Topic				
Indicator Code		BAR.SEC.CMPT.4549.ZS	BAR.SEC.CMPT.5054.FE.Z
Year	Region	Country Name		
1990	East Asia & Pacific	Australia	53.6	49
		Brunei Darussalam	11.7	7
		Cambodia	0.6	0
		China	5.9	3
		Fiji	13.2	7

Topic				
Indicator Code		BAR.SEC.ICMP.75UP.ZS	BAR.SEC.SCHL.1519	BAR
Year	Region	Country Name		
1990	East Asia & Pacific	Australia	77.1	3.7	
		Brunei Darussalam	5.5	2.2	
		Cambodia	2.1	0.8	
		China	5.0	2.0	

```
                          Fiji                                10.9              2.3

Topic
Indicator Code                              UIS.AFR.TRNTP.1.PU.F UIS.AFR.TRNTP.1.PU.M U
Year Region              Country Name
1990 East Asia & Pacific Australia                           nan                  nan
                         Brunei Darussalam                   nan                  nan
                         Cambodia                            nan                  nan
                         China                               nan                  nan
                         Fiji                                nan                  nan

Topic
Indicator Code                              SE.TER.GRAD.AG.FE.ZS SE.TER.GRAD.AG.ZS SE.T
Year Region              Country Name
1990 East Asia & Pacific Australia                           nan               nan
                         Brunei Darussalam                   nan               nan
                         Cambodia                            nan               nan
                         China                               nan               nan
                         Fiji                                nan               nan

Topic
Indicator Code                              UIS.FGP.56.F400500 UIS.FGP.56.FNON400500 UI
Year Region              Country Name
1990 East Asia & Pacific Australia                        nan                   nan
                         Brunei Darussalam                nan                   nan
                         Cambodia                         nan                   nan
                         China                            nan                   nan
                         Fiji                             nan                   nan

Topic
Indicator Code                              UIS.MS.56.F UIS.MS.56.T UIS.MSEP.56 UIS.MSE
Year Region              Country Name
1990 East Asia & Pacific Australia                 nan         nan         nan
                         Brunei Darussalam         nan         nan         nan
                         Cambodia                  nan         nan         nan
                         China                     nan         nan         nan
                         Fiji                      nan         nan         nan

[5 rows x 2310 columns]
```

In [86]:
```
#data_c_mod[('Secondary', 'UIS.DR.2.GPV.G4.F')]
mi_col = data_c_mod.columns
mi_col.get_level_values(0) # Tous les topics des indicateurs
mi_col.get_level_values(1) # Tous les indicateurs
```

Out[86]:
```
Index(['BAR.NOED.1519.FE.ZS', 'BAR.NOED.1519.ZS', 'BAR.NOED.15UP.FE.ZS', 'BAR.NOED.15U
       ...
       'UIS.TEP.7.F', 'UIS.TEP.7.M', 'UIS.TEP.8', 'UIS.TEP.8.F', 'UIS.TEP.8.M', 'UIS.T
```

```
In [87]: data_c_mod.to_csv('../DONNEES/data_c_mod.csv')

In [88]: # Pas terrible l'enregistrement d'un fichier csv avec des multiindex...
         # test = pd.read_csv('../DONNEES/data_c_mod.csv')
         # test.head

In [89]: print("Nombre de valeurs d'indicateurs non vides avant modification :\n{}, pour une ta
               .format(np.sum(~np.isnan(data_c[li_annees_sel].values)),data_c.shape,data_c.shap
         print("Nombre de valeurs d'indicateurs non vides après modification :\n{}, pour une ta
               .format(np.sum(~np.isnan(data_c_mod.values)),data_c_mod.shape,data_c_mod.shape[0
```

```
Nombre de valeurs d'indicateurs non vides avant modification :
2853128, pour une taille de tableau de (227520, 32) = 7280640 cases
Nombre de valeurs d'indicateurs non vides après modification :
2853128, pour une taille de tableau de (5148, 2310) = 11891880 cases
```

1.3.2 Corrélation entre indicateurs d'un même topic

On veut évaluer la corrélation de différents indicateurs, mais il faut comparer leurs valeurs pour les mêmes combinaisons pays-année, qui ne sont pas toujours disponibles.

Dans le tableau ci-dessus, le nombre de données communes (renseignées pour les deux indicateurs comparés) est affiché dans les cases, et la couleur indique le degré de corrélation.

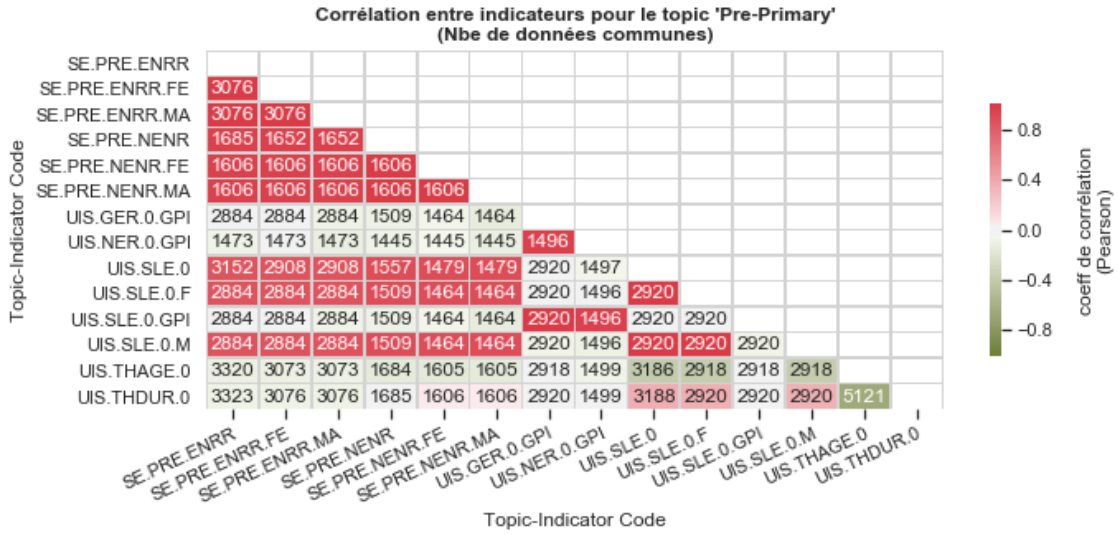
Ces tableaux pourront nous permettre d'identifier les indicateurs fortement corrélés, c'est-à-dire dont les tendances au cours des années sont similaires pour les différents pays.

```
In [90]: def impr_tab_corr(nom_topic,val_min,val_max,tuple_width_length):
             # indicateurs du topic choisi pour toutes les années, tous les pays
             echant = data_c_mod.loc[idx[li_annees_sel,:,:], idx[nom_topic,:]] # ['1990','1995
             echant = echant[echant.columns[val_min:val_max]]
             # liste des indicateurs
             li_indic = [c2 for c1,c2 in echant.columns]
             li_indic2 = [(c1,c2) for c1,c2 in echant.columns]
             # Nombre de valeurs par colonne
             li_nb_val = [echant[tucol].count() for tucol in echant.columns]
             # Nombre de valeurs renseignées communes pour chaque combinaison de deux indicate
             echant_b = ~np.isnan(echant) # valeur 'null' ou pas
             nb_val_comm = np.array([[np.sum(echant_b[j]&echant_b[i]) for i in li_indic2]for j
             my_corr = echant.corr()
             # affichage tableau
             sns.set(font_scale=1.2)
             sns.set(style="whitegrid")
             fig, ax = plt.subplots(figsize = tuple_width_length)
             mask = np.zeros_like(my_corr, dtype=np.bool) # Generate a mask for the upper tria
             mask[np.triu_indices_from(mask)] = True
             palette = sns.diverging_palette(100, 10, as_cmap=True) # palette divergente sur m
             #li_indic = [t2 for t1,t2 in my_corr.columns.values]
             ax = sns.heatmap(my_corr, mask=mask, cmap=palette, vmin=-1, vmax=1, center=0,
                              annot = nb_val_comm, fmt = '',  # '.2f'
```

```
                square = False, linewidths=.5, linecolor = 'lightgrey',
                cbar_kws = {"shrink": .7, 'label': '\ncoeff de corrélation\n(Pearson)
                xticklabels = li_indic, yticklabels = li_indic)
        ax.tick_params(top=False, bottom=True, labeltop=False, labelbottom=True)
        plt.setp(ax.get_xticklabels(), rotation=25, ha="right",rotation_mode="anchor")
        ax.set_title("Corrélation entre indicateurs pour le topic '" + nom_topic + "'\n(NI
        plt.show()
```

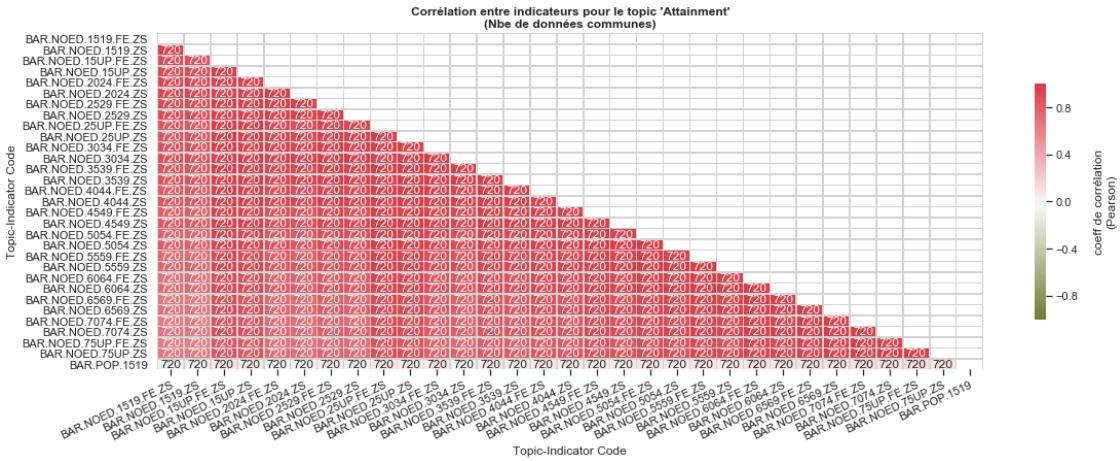
- Topic 'Pre-Primary'

```
In [91]: impr_tab_corr('Pre-Primary',0,14, (10,4))
```

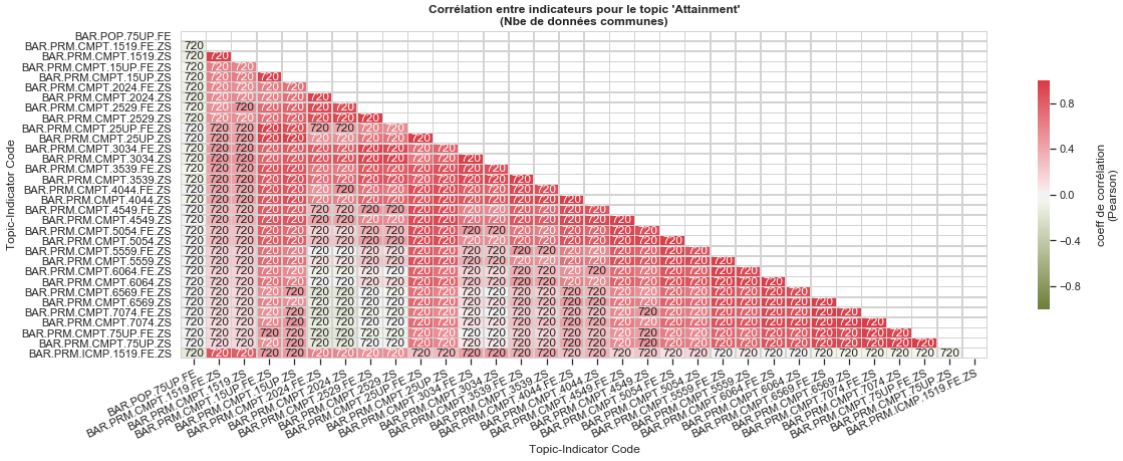


- Topic 'Attainment'

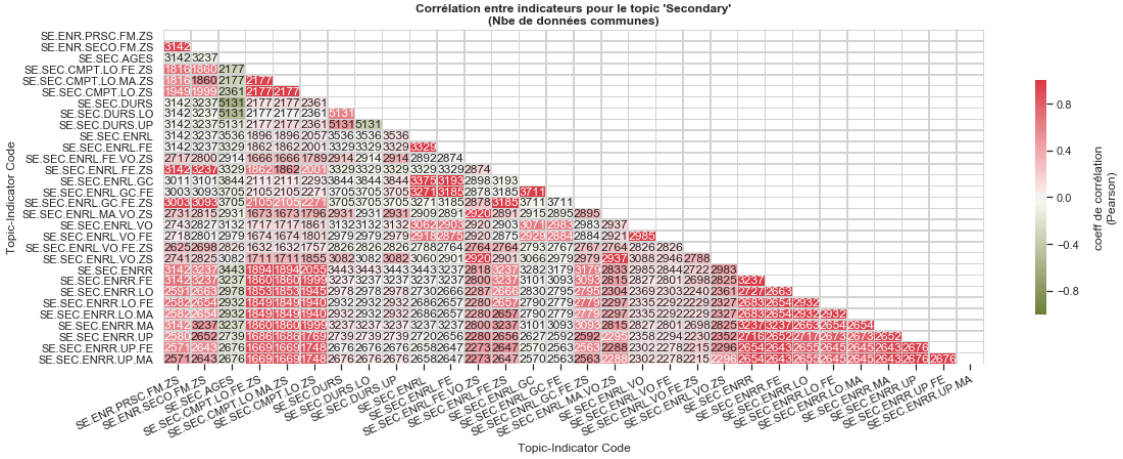
```
In [92]: impr_tab_corr('Attainment',0,31, (18,6))
```



In [93]: `impr_tab_corr('Attainment',59,91, (18,6))`



In [94]: `impr_tab_corr('Secondary',0,30, (18,6))`



1.3.3 Détection des outliers

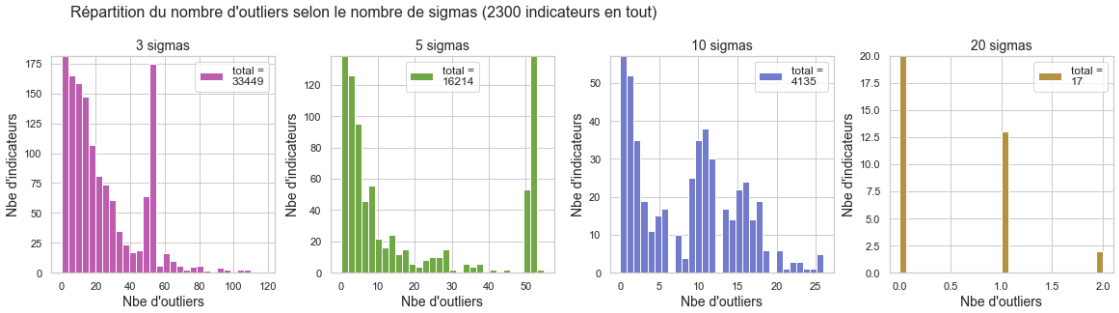
Evaluation du nombre d'outliers selon le seuil Avant d'éliminer ou de remplacer les outliers, on voudrait en connaître le nombre approximatif en fonction du nombre de sigma choisi pour le seuil. On trace les histogrammes du nombre d'outliers par indicateur pour un seuil de 3, 5, 10 et 20 sigmas :

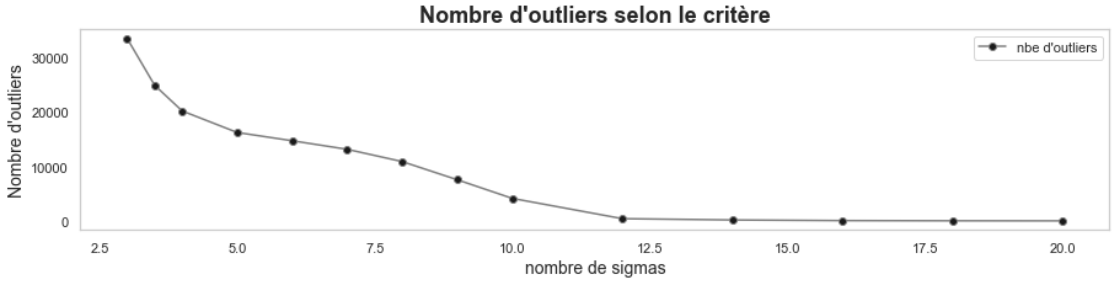
In [95]:
```
# Prend un tableau de valeurs, calcule la moyenne, l'écart type
# et renvoie les valeurs au-delà ou en-deça de x fois l'écart-type

# def detOutliers_col(df,col,n):
#     moy = df[col].mean()
#     std = df[col].std()
#     out_val = [val for val in df[col].values if ((val<moy-(n*std)) or (val>moy+(n*s
#     return df[df[col].isin(out_val)].index, out_val # renvoie les index de la ligne

def detOutliers_df(df_val,n): # prend en entrée une dataframe de toutes les valeurs d
    moy = np.nanmean(df_val)
    std = np.nanstd(df_val)
    v_min = moy-(n*std)
    v_max = moy+(n*std)
    df_flat = df_val.flatten()[~np.isnan(df_val.flatten())]
    out_val = [val for val in df_flat if (val<v_min or val>v_max)]
    # index et colonne remplissant la condition, valeur correspondante... à faire
    return out_val # renvoie les outliers
```

In [96]:
```
#Détermination du nombre de sigmas à prendre en compte pour la détection des outliers
li_ind_uni = data_c['Indicator Code'].unique()
tab_df = [data_c[data_c['Indicator Code']==ind][li_annees_sel] for ind in li_ind_uni]

# tableau du nombre d'outliers pour tous les indicateurs avec n sigma
nb_std = [3,5,10,20]
tab_nb_out = [[len(detOutliers_df(df.values,n)) for df in tab_df] for n in nb_std]
```

In [97]:
```
# Nombre d'outliers par indicateur pour un nombre n d'écart-type
sns.set_style("whitegrid")
fig = plt.figure(figsize = (18,8))

tab_plot = []
for i in range(len(tab_nb_out)):
    tab_plot.append(plt.subplot(1,4,i+1))
    tab_plot[i].hist(tab_nb_out[i], bins = 30, color = colors[i],\
                     label = "total =\n"+str(sum(tab_nb_out[i])))
    plt.xlabel("Nbe d'outliers", fontsize=14), plt.ylabel("Nbe d'indicateurs", fontsi
    #plt.xlim(0, 120)
    y_lim = (np.histogram(tab_nb_out[i], bins = 30)[0][1]*1.1)
    my_y_lim = y_lim if y_lim != 0 else 20
    plt.ylim(0, my_y_lim)
    #plt.text(0,my_y_lim*0.05, "total = "+str(sum(tab_nb_out[i])), fontsize=14)
    plt.title(str(nb_std[i])+" sigmas", fontsize=14)
    plt.grid(True), plt.legend(fontsize=12)

plt.gcf().subplots_adjust(left = 0.25, bottom = 0.5, right = 1.1, top = 0.89, wspace =
fig.suptitle("Répartition du nombre d'outliers selon le nombre de sigmas (2300 indica
             fontsize=16)
```

```
plt.show()

#-----------------------------

fig = plt.figure(figsize = (15,3))
plot5 = plt.subplot(1,1,1)
# tableau du nombre d'outliers pour tous les indicateurs avec n sigma
nb_std = [3,3.5,4,5,6,7,8,9,10,12,14,16,18,20]
nb_out_std = [sum([len(detOutliers_df(df.values,n)) for df in tab_df]) for n in nb_std
plot5.plot (nb_std, nb_out_std, '-o', label = "nbe d'outliers", color = 'grey', marker
#plot5.set_xlim(1980,2018), #plot5.set_ylim(75,220)
plot5.set_xlabel("nombre de sigmas", fontsize = 14), plot5.set_ylabel("Nombre d'outlie
plot5.set_title("Nombre d'outliers selon le critère", fontsize = 18, fontweight = 'bo
plot5.legend(loc = 'upper right'), plt.grid()
plt.show()
```





Elimination des outliers On souhaite éliminer les valeurs aberrantes (outliers) au-delà et en deça de 3 sigmas. On calcule le zscore de tous les indicateurs en se basant sur l'ensemble des valeurs qu'il peut prendre selon les années ([1995;2015]) et les pays (214 pays sélectionnés).

In [98]:
```
# calcule le zscore colonne par colonne, renvoie les z-scores ligne par ligne
data_c_mod_z = data_c_mod.copy(deep=True)
```

```
In [99]: # Calcul pour chaque indicateur du z-score et remplissage d'une nouvelle dataframe
         for (c1,c2) in data_c_mod.columns :
             data_c_mod_z[(c1,c2)] = (data_c_mod[(c1,c2)] - data_c_mod[(c1,c2)].mean()) / data_

In [100]: # crée une nouvelle dataframe sans outliers
          data_c_mod_wo_out = data_c_mod.copy(deep=True)
          #np.warnings.filterwarnings('ignore')
          data_c_mod_wo_out = data_c_mod.where(np.abs(data_c_mod_z.values) < 3)

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: RuntimeWarning: invalid val
  after removing the cwd from sys.path.


In [101]: # comparaison du nombre des valeurs qui ne sont pas 'null' dans les dataframes donné
          print("nb d'outliers retirés : {}"\
                .format(sum(sum(~np.isnan(data_c_mod_z.values)))-sum(sum(~np.isnan(data_c_mod_w
          print("nb de nan crées par le calc du zscore : {}"\
                .format(sum(sum(~np.isnan(data_c_mod.values)))-sum(sum(~np.isnan(data_c_mod_z.v
          sum(sum(~np.isnan(data_c_mod.values)))

nb d'outliers retirés : 33449
nb de nan crées par le calc du zscore : 36


Out[101]: 2853128
```

Problème : apparemment, il existe des colonnes pour lesquelles l'écart-type est nul, ce qui renvoie un z-score null. Il y a 36 données qui sont dans ces colonnes.

```
In [102]: # y a-t-il des lignes pour lesquelles le calcul du z-score donne lieu à la création
          for j in range(data_c_mod.shape[1]): # colonnes
              for i in range(data_c_mod.shape[0]): # lignes
                  if (~np.isnan(data_c_mod.values[i][j])) ^ ~(np.isnan(data_c_mod_z.values[i][j
                      print("col = "+str(j)+" -- ind = "+str(i)+"  :  ",data_c_mod.values[i][j]
                  else:
                      pass

col = 2105 -- ind = 3487  :   100.0  , zscore :   nan
col = 2105 -- ind = 3685  :   100.0  , zscore :   nan
col = 2105 -- ind = 3883  :   100.0  , zscore :   nan
col = 2105 -- ind = 4279  :   100.0  , zscore :   nan
col = 2105 -- ind = 4477  :   100.0  , zscore :   nan
col = 2105 -- ind = 4873  :   100.0  , zscore :   nan
col = 2106 -- ind = 3487  :   100.0  , zscore :   nan
col = 2106 -- ind = 3685  :   100.0  , zscore :   nan
col = 2106 -- ind = 3883  :   100.0  , zscore :   nan
col = 2106 -- ind = 4279  :   100.0  , zscore :   nan
col = 2106 -- ind = 4477  :   100.0  , zscore :   nan
col = 2106 -- ind = 4873  :   100.0  , zscore :   nan
```

```
col = 2107 -- ind = 3487  :   100.0  , zscore :   nan
col = 2107 -- ind = 3685  :   100.0  , zscore :   nan
col = 2107 -- ind = 3883  :   100.0  , zscore :   nan
col = 2107 -- ind = 4279  :   100.0  , zscore :   nan
col = 2107 -- ind = 4477  :   100.0  , zscore :   nan
col = 2107 -- ind = 4487  :   100.0  , zscore :   nan
col = 2107 -- ind = 4685  :   100.0  , zscore :   nan
col = 2107 -- ind = 4873  :   100.0  , zscore :   nan
col = 2107 -- ind = 4883  :   100.0  , zscore :   nan
col = 2108 -- ind = 3487  :   100.0  , zscore :   nan
col = 2108 -- ind = 3685  :   100.0  , zscore :   nan
col = 2108 -- ind = 4279  :   100.0  , zscore :   nan
col = 2108 -- ind = 4873  :   100.0  , zscore :   nan
col = 2109 -- ind = 3487  :   100.0  , zscore :   nan
col = 2109 -- ind = 3685  :   100.0  , zscore :   nan
col = 2109 -- ind = 4279  :   100.0  , zscore :   nan
col = 2109 -- ind = 4873  :   100.0  , zscore :   nan
col = 2110 -- ind = 3487  :   100.0  , zscore :   nan
col = 2110 -- ind = 3685  :   100.0  , zscore :   nan
col = 2110 -- ind = 4279  :   100.0  , zscore :   nan
col = 2110 -- ind = 4487  :   100.0  , zscore :   nan
col = 2110 -- ind = 4685  :   100.0  , zscore :   nan
col = 2110 -- ind = 4873  :   100.0  , zscore :   nan
col = 2110 -- ind = 4883  :   100.0  , zscore :   nan
```

On élimine les indicateurs pour lesquels, après la suppression des outliers, il n'y a aucune donnée 'non null'.

```
In [103]: data_c_mod_wo_out.dropna(how = 'all', axis = 'columns', inplace = False).shape, data_

Out[103]: ((5148, 2304), (5148, 2310))

In [104]: data_c_mod_wo_out.dropna(how = 'all', axis = 'columns', inplace = True)
```

1.3.4 Evaluation du degré de distinctivité des indicateurs entre pays

On souhaite pouvoir quantifier la capacité de chaque indicateur à mettre en évidence des différences entre les pays qui soient éventuellement modélisables.

On cherche par un test statistique à vérifier l'hypothèse nulle d'égalité des moyennes des valeurs des différents pays. On réalise pour cela une ANOVA à un facteur (one-way ANOVA) suivie d'un test de Fisher. Pour chaque indicateur, on divise les données de chaque indicateur en classes regroupant les valeurs de chaque pays pour différentes années. On compare ensuite la variance de l'ensemble des données et la moyenne pondérée des variances des différentes classes en calculant un ratio F :

```
_F = Vinter/Vintra = (SCE/DDL_E)/(SCR/DDL_R),
où F = Vinter/Vintra = (SCE/DDL_E)/(SCR/DDL_R)
avec SCE : somme des écarts des carrés expliquée ou interclasse
```

et SCR : somme des écarts des carrés résiduelle ou intraclasse
et DDL_E et DDL_R les degrés de liberté respectifs._

Le rapport F suit une loi de Fisher, on peut donc obtenir, à partir des degrés de liberté du numérateur et du dénominateur, la p-valeur correspondant à la valeur de F obtenue, c'est-à-dire le pourcentage de chance pour que, avec une telle répartition de nos données, les moyennes soient vraiment distinctes. Plus F sera grand, moins les chances (p-valeurs) seront grandes, et plus F sera proche de 1 ,plus les chances seront grandes.

*** Limitations de la démarche :**

Pour appliquer ce test, il est cependant nécessaire que : - les échantillons soient prélevés aléatoirement et indépendamment dans les populations - le paramètre étudié suive une distribution normale - les variances des populations des classes soient toutes égales entre elles (homoscédasticité)

Or ici les données *ne sont pas prélevées aléatoirement*, et ne suivent *sans doute pas une distribution normale*. Elles dépendent en fait du paramètre année et sont susceptibles de suivre des tendances variables, ce qui se traduira par des *variances différentes*.

*** Différentes approches pour la sélection des données :**

Le test de Fisher et l'ANOVA peuvent être calculés avec des classes ayant des tailles d'échantillon différentes. Cependant, le fait que chaque valeur dans les classes corresponde à une année, il est probable que, la manière dont on sélectionnera les données pour le calcul de F influencera les résultats. Deux approches ont été adoptées pour la sélection : - on effectue le calcul en tenant compte de toutes les valeurs (année/pays) disponible - on sélectionne seulement les pays pour lesquels les données (années) renseignées sont communes.

Le rapport F n'est pas calculé s'il y a moins de 3 pays en tout ou si le nbe total de valeurs est inférieur à 2 fois le nombre des pays.

Trois fonctions sont définies ci-dessous : - "calc_Fisher_scipy" : calcul en utilisant la fonction f_oneway de scipy (deux options de sélection de données possibles. - "calc_Fisher_all" : calcul de F en appliquant la formule ci-dessus, et en utilisant toutes les données. - "calc_Fisher_bis" : calcul de F en appliquant la formule ci-dessus (deux options de sélection de données possibles).

```
In [105]: def calc_Fisher_scipy (df, option): # sur une dataframe regroupant les données d'un

                if option == 'opt': # on conserve toutes les données
                    df_mod = df.unstack('Year')
                    df_mod = df_mod.T
                    df_mod.columns = df_mod.columns.droplevel()
                    df_mod.dropna(how= 'all', axis = 1, inplace = True)
                elif option == 'opt_bis': # on élimine les années vides, puis les pays n'ayant p
                    df_mod = df.unstack('Year')
                    df_mod = df_mod.dropna(how = 'all', axis = 1)
                    df_mod = df_mod.dropna(how = 'any', axis = 0)
                    df_mod = df_mod.T
                    df_mod.columns = df_mod.columns.droplevel()

                ### Vérif : ANOVA à un paramètre de la librairie scipy stats [entrée : liste val
                #Pour chaque classe, on garde toutes les données, en ignorant les nan
                if df_mod.columns.size < 3: # pas de calcul s'il n'y a que 1 ou 2 pays
                    return np.nan ,np.nan,sum(sum(~np.isnan(df_mod.values))), df_mod.columns.size
```

```
            else:
                f, p = st.f_oneway(*[df_mod[pays][~df_mod[pays].isna()] for pays in df_mod.co
                return f,p, sum(sum(~np.isnan(df_mod.values))), df_mod.columns.size

In [106]: def calc_Fisher_all (df): # sur une dataframe regroupant les données d'un seul indic

            MOY = df.mean() # moyenne de toutes les valeurs (années et pays)
            Pays_moy = df.groupby(level="Country Name").mean() # moyenne des valeurs par pays
            Pays_count = df.groupby(level="Country Name").count() # nbe des valeurs par pays

            # Coeff de Fisher
            # non calculé s'il y a - de 3 pays en tout ou si le nbe tot de valeurs < 3 fois
            if (len(Pays_moy) < 4) or ((df.count()-len(Pays_moy)) < 2*len(Pays_moy)):
                return  np.nan, sum(~np.isnan(df.values))
            else :
                # Somme des carrés des écarts expliquée = interclasse
                SCE = np.nansum(Pays_count*(np.square(Pays_moy - MOY))) # Somme des carrés d
                DDL_E = len(Pays_moy)-1 # Degré de liberté SCE (nombre de classes (pays) - 1
                # Somme des carrés des écarts résiduelle = intraclasse
                SCR = 0
                for pays in df.index.get_level_values(2).unique(): # pour chaque pays de la
                    SCR += (np.nansum((np.square(df.loc[idx[:,:,pays]]-Pays_moy[pays]))))
                # Degré de liberté SCR (somme des counts(années dispo) de chaque classe (pay
                DDL_R = df.count()-len(Pays_moy)

                return (SCE/DDL_E)/(SCR/DDL_R), sum(~np.isnan(df.values))

In [107]:  def calc_Fisher_bis (df,option): # sur une dataframe regroupant les données d'un se

            if option == 'opt': # on conserve toutes les données
                df_mod = df.unstack('Year')
                df_mod = df_mod.T
                df_mod.columns = df_mod.columns.droplevel()
                df_mod.dropna(how= 'all', axis = 1, inplace = True)
            elif option == 'opt_bis': # on élimine les années vides, puis les pays n'ayant p
                df_mod = df.unstack('Year')
                df_mod = df_mod.dropna(how = 'all', axis = 1)
                df_mod = df_mod.dropna(how = 'any', axis = 0)
                df_mod = df_mod.T
                df_mod.columns = df_mod.columns.droplevel()

            # F-score (non calculé s'il y a moins de 3 pays en tout ou s'il y a moins de 3 a
            if (df_mod.index.size < 4) or (df_mod.columns.size < 4):
                return  np.nan, sum(sum(~np.isnan(df_mod.values)))
            else :
                MOY = np.nanmean(df_mod.values) # moyenne de toutes les uzleurs (années-pays
                Pays_moy = df_mod.mean(axis=1) # moyenne des valeurs (ttes les années) par c
                Pays_count = df_mod.columns.size # nbe des valeurs par pays (nbe d'années)
```

```
                # Somme des carrés des écarts expliquée = interclasse
                SCE = np.sum(Pays_count*(np.square(Pays_moy - MOY)))
                DDL_E = len(Pays_moy)-1 # Degré de liberté SCE (nombre de classes (pays) -1)
                # Somme des carrés des écarts résiduelle = intraclasse
                SCR = 0
                for pays in df_mod.index: # pour chaque pays de la liste d'un indicateur
                    SCR += (np.sum((np.square(df_mod.loc[pays]-Pays_moy[pays]))))
                # Degré de liberté SCR (somme des counts(années dispo) de chaque classe (pay
                DDL_R = df_mod.index.size*df_mod.columns.size - len(Pays_moy)

                return   (SCE/DDL_E)/(SCR/DDL_R), sum(sum(~np.isnan(df_mod.values)))
```

In [108]:
```
### Test sur un indicateur
df = data_c_mod_wo_out[data_c_mod_wo_out.columns[168]]

calc_Fisher_all (df), \
calc_Fisher_bis (df,'opt'), calc_Fisher_bis (df,'opt_bis'),\
calc_Fisher_scipy (df,'opt'), calc_Fisher_scipy (df,'opt_bis')
```

Out[108]:
```
((20.995128411203073, 720),
 (13.195514221656703, 720),
 (15.859993054875845, 720),
 (31.915430312086862, 3.364049144378911e-202, 720, 144),
 (31.915430312086862, 3.364049144378911e-202, 720, 144))
```

In [109]:
```
# Création d'une dataframe réunissant les scores et le nbe de données sur lesquelles
Fisher1 = np.array([calc_Fisher_all(data_c_mod_wo_out[c1,c2]) for c1,c2 in data_c_mod
Fisher2 = np.array([calc_Fisher_bis(data_c_mod_wo_out[c1,c2],'opt') for c1,c2 in data
Fisher2_bis = np.array([calc_Fisher_bis(data_c_mod_wo_out[c1,c2],'opt_bis') for c1,c
Fisher3 = np.array([calc_Fisher_scipy(data_c_mod_wo_out[c1,c2],'opt') for c1,c2 in da
Fisher3_bis = np.array([calc_Fisher_scipy(data_c_mod_wo_out[c1,c2],'opt_bis') for c1
```

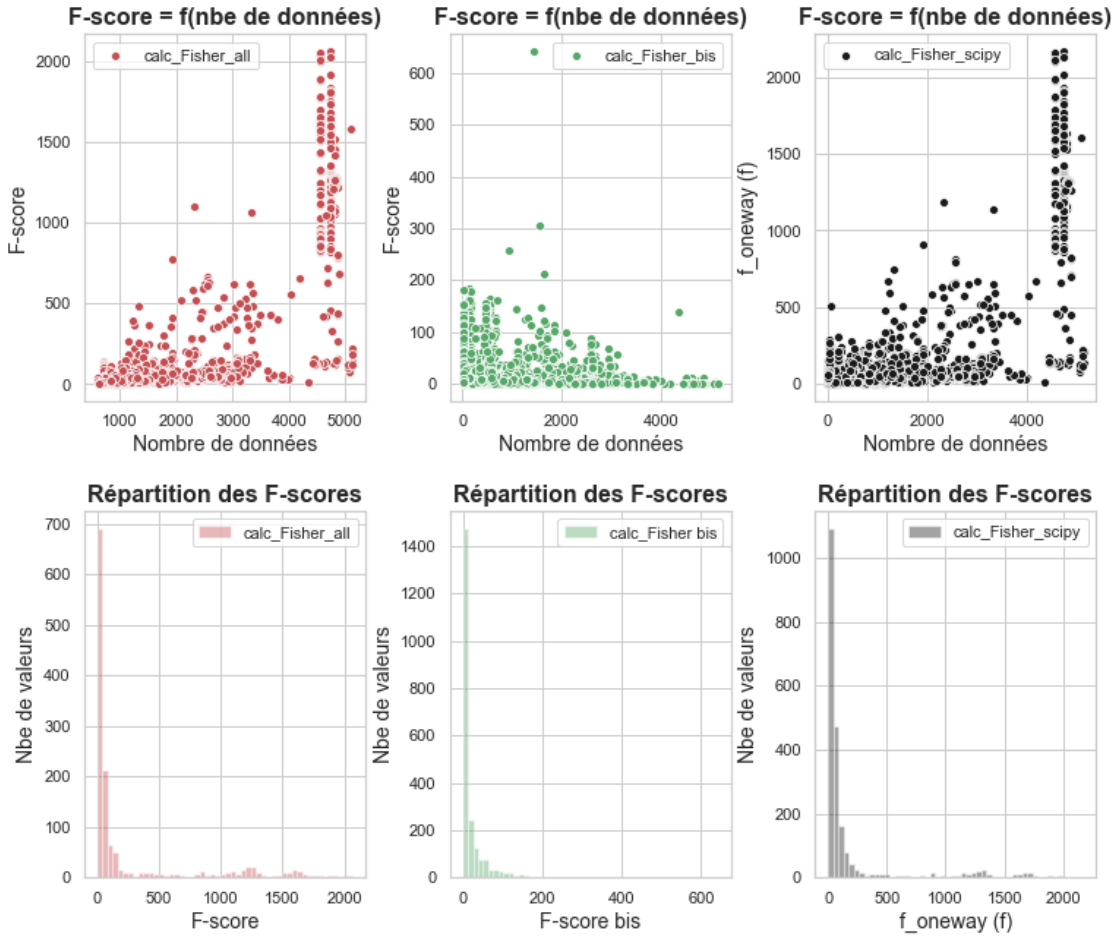
In [110]:
```
df_F_score = pd.DataFrame(data = {
                                  "Topic" : data_c_mod_wo_out.columns.get_level_values(0),
                                  "Indicator Name"  : data_c_mod_wo_out.columns.get_level_
                                  "calc_Fisher_scipy opt (f)" : Fisher3[:,0],
                                  "calc_Fisher_scipy opt (p-val)" : Fisher3[:,1],
                                  "calc_Fisher_scipy opt (nb)" : Fisher3[:,2],
                                  "calc_Fisher_scipy opt (nb pays)" : Fisher3[:,3],
                                  "calc_Fisher_scipy opt_bis (f)" : Fisher3_bis[:,0],
                                  "calc_Fisher_scipy opt_bis (p-val)" : Fisher3_bis[:,1],
                                  "calc_Fisher_scipy opt_bis (nb)" : Fisher3_bis[:,2],
                                  "calc_Fisher_all (f)" : Fisher1[:,0],
                                  "calc_Fisher_all (nb)" : Fisher1[:,1],
                                  "calc_Fisher bis opt (f)" : Fisher2[:,0],
                                  "calc_Fisher bis opt (nb)" : Fisher2[:,1],
                                  "calc_Fisher bis opt_bis (f)" : Fisher2_bis[:,0],
                                  "calc_Fisher bis opt_bis (nb)" : Fisher2_bis[:,1],
                                  })
```

```
df_F_score.set_index(["Topic", "Indicator Name"], inplace = True)
# Dataframe des indicateurs, classés par la valeur de F et par topic
df_F_score.sort_values(by=['Topic','Indicator Name'], inplace = True)
```

In [111]:
```
# Affichage des résultats
fig = plt.figure(figsize = (16,8))

xy1 = df_F_score[["calc_Fisher_all (nb)", "calc_Fisher_all (f)"]].dropna(how = 'any'
xy2 = df_F_score[["calc_Fisher bis opt (nb)", "calc_Fisher bis opt (f)"]].dropna(how
xy3 = df_F_score[["calc_Fisher_scipy opt (nb)","calc_Fisher_scipy opt (nb pays)",\
                  "calc_Fisher_scipy opt (f)"]].dropna(how = 'any', inplace = False)

ax1 = plt.subplot(2,3,1)
ax1 = sns.scatterplot(xy1["calc_Fisher_all (nb)"], xy1["calc_Fisher_all (f)"] , colo
ax1.set_title("F-score = f(nbe de données)",  fontsize = 16, fontweight = 'bold')
ax1.set_xlabel("Nombre de données", fontsize = 14)
ax1.set_ylabel("F-score", fontsize = 14)

ax2 = plt.subplot(2,3,2)
ax2 = sns.scatterplot(xy2["calc_Fisher bis opt (nb)"], xy2["calc_Fisher bis opt (f)"]
ax2.set_title("F-score = f(nbe de données)",  fontsize = 16, fontweight = 'bold')
ax2.set_xlabel("Nombre de données", fontsize = 14)
ax2.set_ylabel("F-score", fontsize = 14)

ax3 = plt.subplot(2,3,3)
ax3 = sns.scatterplot(xy3["calc_Fisher_scipy opt (nb)"], xy3["calc_Fisher_scipy opt
ax3.set_title("F-score = f(nbe de données)",  fontsize = 16, fontweight = 'bold')
ax3.set_xlabel("Nombre de données", fontsize = 14)
ax3.set_ylabel("f_oneway (f)", fontsize = 14)

ax4 = plt.subplot(2,3,4)
ax4 = sns.distplot(xy1["calc_Fisher_all (f)"].values, color = 'r', label = "calc_Fis
ax4.set_title("Répartition des F-scores",  fontsize = 16, fontweight = 'bold')
ax4.set_xlabel("F-score", fontsize = 14)
ax4.set_ylabel("Nbe de valeurs", fontsize = 14)
ax4.legend()

ax5 = plt.subplot(2,3,5)
ax5 = sns.distplot(xy2["calc_Fisher bis opt (f)"], color = 'g', label = "calc_Fisher
ax5.set_title("Répartition des F-scores",  fontsize = 16, fontweight = 'bold')
ax5.set_xlabel("F-score bis", fontsize = 14)
ax5.set_ylabel("Nbe de valeurs", fontsize = 14)
ax5.legend()

ax6 = plt.subplot(2,3,6)
ax6 = sns.distplot(xy3["calc_Fisher_scipy opt (f)"], color = 'k', label = "calc_Fish
ax6.set_title("Répartition des F-scores",  fontsize = 16, fontweight = 'bold')
ax6.set_xlabel("f_oneway (f)", fontsize = 14)
```

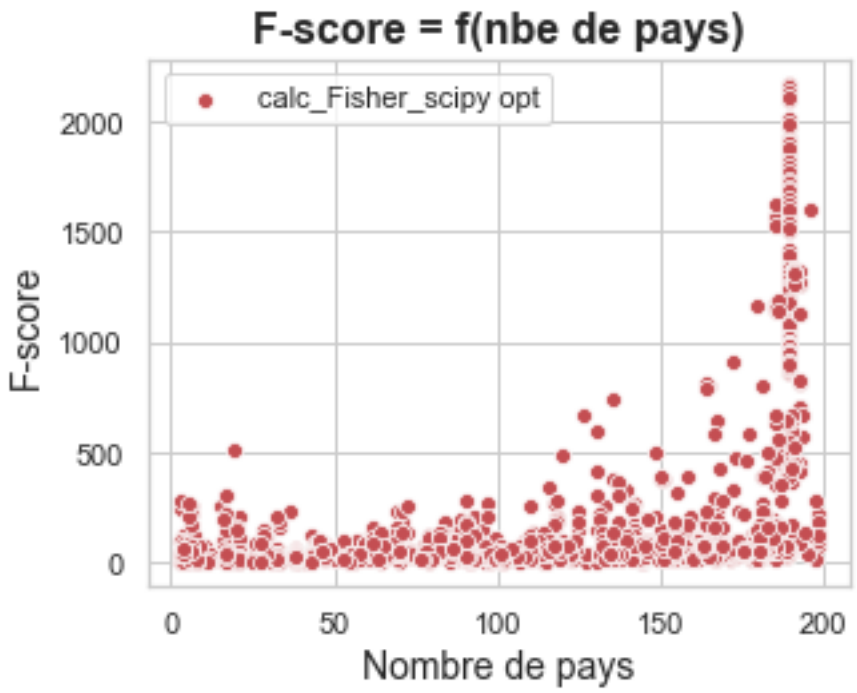
```
        ax6.set_ylabel("Nbe de valeurs", fontsize = 14)
        ax6.legend()

        plt.gcf().subplots_adjust(left = 0.1, bottom = 0.2, right = 0.7, top = 1.2, wspace =

        plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a no
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

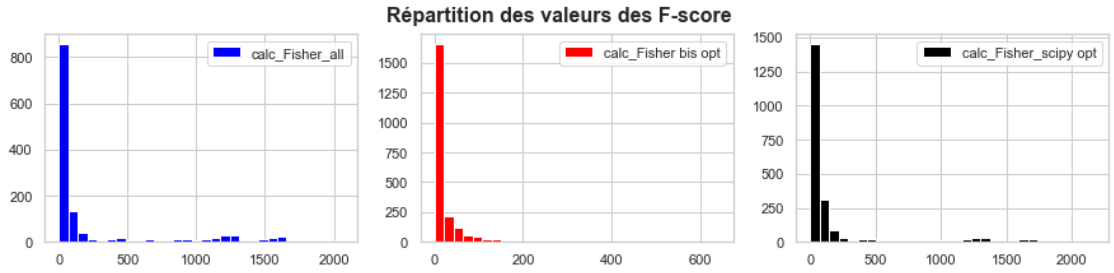



```
In [112]: fig = plt.figure(figsize = (16,8))

        ax1 = plt.subplot(2,3,1)
        ax1 = sns.scatterplot(xy3["calc_Fisher_scipy opt (nb pays)"], xy3["calc_Fisher_scipy
        ax1.set_title("F-score = f(nbe de pays)",  fontsize = 16, fontweight = 'bold')
        ax1.set_xlabel("Nombre de pays", fontsize = 14)
        ax1.set_ylabel("F-score", fontsize = 14)
```

Out[112]: Text(0,0.5,'F-score')



In [113]:
```
# répartition des scores
y1 = df_F_score["calc_Fisher_all (f)"][~np.isnan(df_F_score["calc_Fisher_all (f)"])]
y2 = df_F_score["calc_Fisher bis opt (f)"][~np.isnan(df_F_score["calc_Fisher bis opt
y3 = df_F_score["calc_Fisher_scipy opt (f)"][~np.isnan(df_F_score["calc_Fisher_scipy

fig = plt.figure(figsize = (15,3))
fig.suptitle('Répartition des valeurs des F-score', fontsize=16, fontweight = 'bold'

ax1 = plt.subplot(1,3,1)
ax1.hist(y1, bins=30, color = 'blue', label = "calc_Fisher_all")
ax1.legend()

ax2 = plt.subplot(1,3,2)
ax2.hist(y2, bins=30, color = 'red', label = "calc_Fisher bis opt")
ax2.legend()

ax3 = plt.subplot(1,3,3)
ax3.hist(y3, bins=30, color = 'black', label = "calc_Fisher_scipy opt")
ax3.legend()
plt.show()
```




```
In [114]: # df_F_score2 = pd.DataFrame(data = {
          #                             "Topic" : data_c_mod_wo_out.columns.get_level_values(0
          #                             "Indicator Name"  : data_c_mod_wo_out.columns.get_leve
          #                             })
          # df_F_score2.set_index(["Topic", "Indicator Name"], inplace = True)
          # df_F_score2.sort_values(by=['Topic','Indicator Name'], inplace = True)
          # # On teste l'égalité des moyennes comme hypothèse nulle
          # # y a-t-il des indics tels que f est très petit, et que la p-valeur recommence à d
          # df_F_score2[df_F_score2["calc_Fisher_scipy opt(p-val)"]>0.001].dropna(how="any", i
```

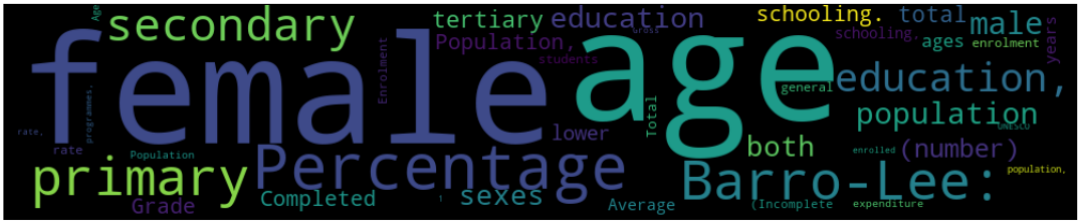
1.4 3. Tracé de certains indicateurs pertinents

Il nous faut maintenant effectuer un choix des indicateurs pertinents à tracer. Les critères à reenir sont les suivants : - être pertinent du point de vue de notre problématique - avoir suffisamment de valeurs disponibles (années et pays) - permettre de disinguer les pays entre eux

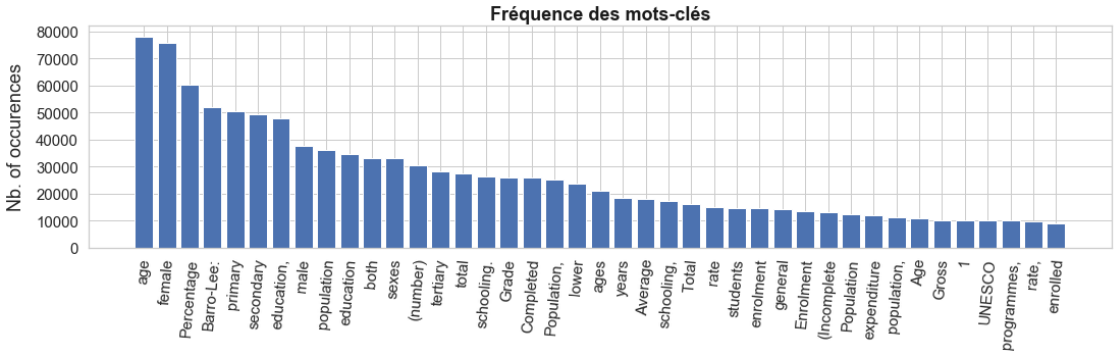
1.4.1 3.0 Overview rapide des indicateurs : exploration des mots clés dans les noms d'indicateurs

Afin d'avoir une première idée du contenu des indicateurs, traçons un nuage de mots-clés des noms des indicateurs :

```
In [115]: list_pop = ["in", "of", "a", "and", "Per", "by", "the", "with", "to", "from",\
                      "for", "who", "on", "are"] # liste des mots non considérés

          # génération dictionnaire effectif mots-clés
          dic_occs = enum_mots_cmpt(data_c["Indicator Name"], 50) # à partir des noms d'indica
          dic_occs = filt_dict(dic_occs,list_pop) # filtrage des mots indésirables

          nuageMots(dic_occs)
```



```
In [116]: histMots(dic_occs)
```



1.4.2 3.1 Analyse de la problématique

Voici les sujets sur lesquels on cherche des informations :

- **public visé** :
  - Population ayant terminé le collège (lower secondary schooling)
  - Population actuellement au lycée (upper secondary schooling)
  - Population déscolarisée, en âge d'être au lycée
  - Population actuellement à l'université ou dans des filières post-bac (tertiary schooling, post secondary/non tertiary)
- **besoins en matière d'éducation** :
  - nbe d'enseignants par élève au lycée, à l'université
  - taux d'achèvement du cycle secondaire
- **possibilités de déploiement dans le pays** :
  - accès à l'internet
  - accès à un ordinateur personnel
- **possibilités de financement** :
  - part du budget familial consacré à l'éducation
  - pouvoir d'achat moyen

1.4.3 3.2 Sélection grossière d'indicateurs pertinents

En balayant la liste des indicateurs disponibles, topic par topic, on réalise une première sélection des indicateurs pertinents à notre problématique :

```
In [117]: # Recherche de mot-clés dans les indicateurs
          # attention aux valeurs null dans les 'short definitions'
          my_list = series["Short definition"].dropna(inplace = False)
          tab = []
          for i in my_list.index:
              if ('household' in my_list.loc[i] and \
                  'funding' in my_list.loc[i] and\
                  'GDP' in my_list.loc[i] and\
                  'capita'in my_list.loc[i]) :
                  tab.append((i, series["Series Code"][i], series["Short definition"][i]))
          tab

Out[117]: [(2381,
            'SE.XPD.TOTL.GD.ZS',
            "Total general (local, regional and central) government expenditure on education (

In [120]: # Recherche de mot-clés dans les indicateurs
          # attention aux valeurs null dans les 'short definitions'
          my_list = series["Short definition"].dropna(inplace = False)
          tab = []
          for i in my_list.index:
              if ('funding' in my_list.loc[i] and \
                  'household' in my_list.loc[i] and\
                  'GDP' in my_list.loc[i] and\
                  'capita'in my_list.loc[i]) :
                  tab.append((i, series["Series Code"][i], series["Short definition"][i]))
          tab

Out[120]: [(2381,
            'SE.XPD.TOTL.GD.ZS',
            "Total general (local, regional and central) government expenditure on education (
```

1. Public visé : *Population ayant terminé le collège (lower secondary schooling)*

- School age population, secondary education SP.SEC.TOTL.IN
- School age population, upper secondary education SP.SEC.UTOT.IN
- Enrolment rate upper secondary UIS.NERA.3
- Enrolment in secondary education (number) SE.SEC.ENRL
- Enrolment in secondary education, general (number) SE.SEC.ENRL.GC
- Enrolment in secondary education, vocational (number) SE.SEC.ENRL.VO
- Enrolment in secondary education, privae institutions (number) UIS.E.23.PR
- Enrolment in upper secondary education (number) UIS.E.3
- Enrolment in upper secondary education, public (number) UIS.E.3.PU
- Enrolment in upper secondary education, private (number) UIS.E.3.PR
- Enrolment in upper secondary education, vocational (number) UIS.E.3.VO
- Enrolment in upper secondary education, general (number) UIS.E.3.GPV
- Gross enrolment rate in secondary education SE.SEC.ENRR
- Gross enrolment rate in upper secondary education SE.SEC.ENRR.UP

- Net enrolment rate, upper secondary UIS.NER.3
- Total net enrolment rate upper secondary UIS.NERT.3

Population actuellement au lycée (upper secondary schooling)

- Barro-Lee percentage of 15-19 with secondary schooling incompled and completed BAR.SEC.ICMP.1519.ZS
- Barro-Lee percentage of 15-19 with secondary schooling completed BAR.SEC.CMPT.1519.ZS
- Barro-Lee percentage of 15-19 with tertiary schooling incompled and completed BAR.TER.ICMP.1519.ZS
- Barro-Lee percentage of 15-19 with tertiary schooling completed BAR.TER.CMPT.1519.ZS
- Barro-Lee percentage of 20-24 with secondary schooling incompled and completed BAR.SEC.ICMP.2024.ZS
- Barro-Lee percentage of 20-24 with secondary schooling completed BAR.SEC.CMPT.2024.ZS
- Barro-Lee percentage of 20-24 with tertiary schooling incompled and completed BAR.TER.ICMP.2024.ZS
- Barro-Lee percentage of 20-24 with tertiary schooling completed BAR.TER.CMPT.2024.ZS

Population ayant obtenu le bac ou un diplôme d'études supérieure

- Percentage of population age 25+ with a completed bachelor's or equivalent degree (ISCED6) UIS.EA.2T6.AG25T99
- Percentage of population age 25+ with a completed bachelor's or equivalent degree (ISCED6) UIS.EA.6.AG25T99
- Percentage of population age 25+ with a completed bachelor's or higher degree (ISCED6) UIS.EA.6T8.AG25T99

Population déscolarisée, en âge d'être au lycée

- Out-of-school youth of upper sc school age (number) UIS.OFST.3.CP
- Rate of out-of-school youth of upper secondary school age UIS.ROFST.3.CP

Population actuellement à l'université ou dans des filières post-bac (tertiary schooling, post secondary/non tertiary)

- School age population, post-secondary non-tertiary UIS.SAP.4
- School age population, tertiary education SP.TER.TOTL.IN
- Gross enrolment ratio post-secondary non-tertiary UIS.GER.4
- Gross enrolment ratio tertiary SE.TER.ENRR
- Enrolment in post-secondary non-tertiary education UIS.E.4
- Total enrolment in tertiary education all programmes (number) SE.TER.ENRL

Informations générales sur la population

- Population totale SP.POP.OTL
- Population de XX ans SP.POP.AGXX.TO.UN
- Population dans la tranche 14-18 ans SP.POP.1418.TO.UN
- Population dans la tranche 15-24 ans SP.POP.1524.TO.UN
- Croissance de la population SP.POP.GROW

2. Possibilités de déploiement dans le pays : *accès à l'internet*

- Internet users per 100 people IT.NET.USER.P2

accès à un ordinateur personnel

- Personal computers per 100 people IT.CMP.PCMP.P2

3. Besoins en matière d'éducation *nbe d'enseignants par élève au lycée, à l'université*

- Teachers in tertiary education programmes (number) SE.TER.TCHR
- Percentage of teachers in upper secondary, qualified UIS.QUTP.3
- Percentage of teachers in upper secondary, trained UIS.TRTP.3
- Percentage of teachers in post-secondary/non tertiary, trained UIS.TRTP.4
- Pupil/teacher ratio in upper secondary UIS.PTRHC.3
- Pupil/qualified teacher ratio in upper secondary UIS.PTRHC.3.QUALIFIED
- Pupil/trained teacher ratio in upper secondary UIS.PTRHC.3.TRAINED
- Pupil/teacher ratio in tertiary UIS.PTRHC.56

taux d'achèvement du cycle secondaire

- DHS : Secondary completion rate HH.DHS.SCR
- DHS : Net attendance rate in secondary HH.DHS.NAR.23
- DHS : Gross attendance rate in post-secondary HH.DHS.GAR.456
- MICS : Secondary completion rate HH.MICS.SCR
- MICS : Net attendance rate in secondary HH.MICS.NAR.23
- MICS : Gross attendance rate in post-secondary HH.MICS.GAR.456

4. Possibilités de financement : *part du budget familial consacré à l'éducation*

- Initial household funding per secondary student as part of GDP per capita UIS.XUNIT.GDPCAP.23.FSHH
- Initial household funding per tertiary student as part of GDP per capita UIS.XUNIT.GDPCAP.5T8.FSHH
- Initial household funding of secondary education as percentage of GDP UIS.XGDP.23.FSHH.FFNTR
- Initial household funding of tertiary education as percentage of GDP UIS.XGDP.5T8.FSHH.FFNTR
- Initial government funding per secondary student as part of GDP per capita UIS.XUNIT.GDPCAP.23.FSGOV
- Initial government funding per upper secondary student as part of GDP per capita UIS.XUNIT.GDPCAP.3.FSGOV
- Initial government funding per tertiary student as part of GDP per capita UIS.XUNIT.GDPCAP.5T8.FSGOV

pouvoir d'achat moyen

- GDP per capita (current US$) NY.GDP.PCAP.CD

1.4.4 3.3 Sélection minimale d'indicateurs à tracer

1. Public visé :

- Enrolment in upper secondary education (number) UIS.E.3
- Enrolment in post-secondary non-tertiary education (number) UIS.E.4
- Population dans la tranche 15-24 ans (number) SP.POP.1524.TO.UN

2. Possibilités de déploiement dans le pays :

- Internet users per 100 people IT.NET.USER.P2

3. Besoins en matière d'éducation

- Pupil/teacher ratio in upper secondary UIS.PTRHC.3
- Pupil/teacher ratio in tertiary UIS.PTRHC.56

4. Possibilités de financement

- GDP per capita (current USD) NY.GDP.PCAP.CD

```
In [121]: data_c_mod_wo_out_bis = data_c_mod.copy(deep = True)
          data_c_mod_wo_out_bis.columns = data_c_mod_wo_out_bis.columns.droplevel()
          data_c_mod_wo_out_bis.head(2)
```

Out[121]:

Indicator Code			BAR.NOED.1519.FE.ZS	BAR.NOED.1519.ZS	BA
Year	Region	Country Name			
1990	East Asia & Pacific	Australia	0.1	0.1	
		Brunei Darussalam	20.1	19.7	

Indicator Code			BAR.POP.6064.FE	BAR.POP.6569	BAR.POP.65
Year	Region	Country Name			
1990	East Asia & Pacific	Australia	364.0	658.0	
		Brunei Darussalam	2.0	3.0	

Indicator Code			BAR.PRM.ICMP.2024.ZS	BAR.PRM.ICMP.2529.I
Year	Region	Country Name		
1990	East Asia & Pacific	Australia	4.1	
		Brunei Darussalam	4.2	

Indicator Code			BAR.PRM.SCHL.5054	BAR.PRM.SCHL.5054.FE
Year	Region	Country Name		
1990	East Asia & Pacific	Australia	5.7	5.7
		Brunei Darussalam	3.6	2.4

Indicator Code			BAR.SEC.CMPT.25UP.ZS	BAR.SEC.CMPT.3034.I
Year	Region	Country Name		
1990	East Asia & Pacific	Australia	52.0	
		Brunei Darussalam	24.1	

```
Indicator Code                                BAR.SEC.ICMP.5054.ZS  BAR.SEC.ICMP.5559.
Year Region              Country Name
1990 East Asia & Pacific Australia                            69.0
                         Brunei Darussalam                    23.7

Indicator Code                                UIS.AFR.GTCTR.1.F  UIS.AFR.GTCTR.1.M  UI
Year Region              Country Name
1990 East Asia & Pacific Australia                          nan                nan
                         Brunei Darussalam                  nan                nan

Indicator Code                                UIS.TRTP.23.GPI  UIS.TRTP.3  UIS.TRTP.3.
Year Region              Country Name
1990 East Asia & Pacific Australia                        nan         nan           na
                         Brunei Darussalam                nan         nan           na

Indicator Code                                SE.TER.PRIV.ZS  SE.TOT.ENRR  UIS.E.4  UI
Year Region              Country Name
1990 East Asia & Pacific Australia                       nan          nan      nan
                         Brunei Darussalam               nan          nan      nan

Indicator Code                                UIS.FOSEP.56.F800.M  UIS.FOSEP.56.FUK  U
Year Region              Country Name
1990 East Asia & Pacific Australia                             nan               nan
                         Brunei Darussalam                     nan               nan

[2 rows x 2310 columns]
```

In [122]:
```
# Vérification de la présence des indicateurs choisis dans la base
li_indic_o = series["Series Code"].unique()
li_indic_o_bis = data_c_mod_wo_out_bis.columns
li_ind_plot = ["UIS.E.3", "UIS.E.4", "SP.POP.1524.TO.UN",\
"IT.NET.USER.P2", "UIS.PTRHC.3", "UIS.PTRHC.56", "NY.GDP.PCAP.CD"]

print("Dans la liste d'origine des séries ?", [ind in li_indic_o for ind in li_ind_pl
print("Dans la liste des séries nettoyées ?", [ind in li_indic_o_bis for ind in li_i
```

```
Dans la liste d'origine des séries ? [True, True, True, True, True, True, True]
Dans la liste des séries nettoyées ? [True, True, True, True, True, True, True]
```

In [123]:
```
# Affichage des F-score des indicateurs choisis
mask = [(code in li_ind_plot) for top,code in df_F_score.index]
df_F_score[mask]
```

Out[123]:
```
                                                                    calc_Fisher_sc
Topic                                                 Indicator Name
Economic Policy & Debt: National accounts: US$ ... NY.GDP.PCAP.CD
Infrastructure: Communications                     IT.NET.USER.P2
```

```
Population                                              SP.POP.1524.TO.UN
Secondary                                               UIS.E.3
Teachers                                                UIS.PTRHC.3
                                                        UIS.PTRHC.56
Tertiary                                                UIS.E.4
```

Remarque : Tous les indicateurs ont des F-score élevés, donc les moyennes des valeurs par pays sont nettement distinctes.

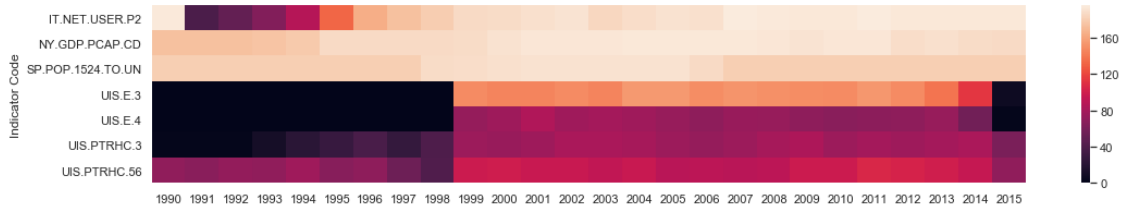
```
In [124]: # Création d'une dataframe restreinte (version multi-index)
          data_plot = data_c_mod_wo_out_bis[li_ind_plot]
          data_plot.head(2)
```

```
Out[124]: Indicator Code                                UIS.E.3  UIS.E.4  SP.POP.1524.TO.UN  IT.I
          Year Region              Country Name
          1990 East Asia & Pacific Australia               nan      nan          2733117.0
                                   Brunei Darussalam       nan      nan            48309.0
```

```
In [125]: # Création d'une dataframe restreinte (version empilée en colonnes)
          data_plot_hm = data_c_mod_wo_out_bis[li_ind_plot].copy(deep = True)
          data_plot_hm = data_plot_hm.unstack('Year').stack('Indicator Code')
          data_plot_hm.reset_index(inplace = True)
          data_plot_hm.columns.names = [""]
          data_plot_hm.head(1)
```

```
Out[125]:              Region Country Name  Indicator Code  1990  1991  1992  1993  1994  19
          0  East Asia & Pacific    Australia  IT.NET.USER.P2   0.6   1.1   1.8   2.0   2.2   2
```

```
In [126]: ### Heatmap du nombre d'indicateurs non nuls (pays/années)
          nb_ind_cnt = data_plot_hm.groupby(['Indicator Code']).count()[li_annees_sel]
          # Tableau des nombres de pays ayant des valeurs dispo pour chaque indicateur et chaq
          fig = plt.figure(figsize = (18,3))
          heat_map = sns.heatmap(nb_ind_cnt)
```



Tous les indicateurs choisis ont un bon taux de renseignement pour les années 1999-2014. Il manque des données pour les ratio élèves/professeurs avant 1999 (UIS.PTRHC.3 & UIS.PTRHC.56), ainsi que pour le nombre d'inscripion au lycée (UIS.E.3) et dans l'enseignement supérieur (UIS.E.4)

In [127]:
```
# Manipulation de la dataframe : empilement complet
data_plot_st = data_plot.copy(deep=True)
#set_index(["Region", "Country Name", "Indicator Code"], inplace = False)
data_plot_st.columns = pd.MultiIndex.from_product([data_plot_st.columns, ['val']], na
data_plot_st.columns = data_plot_st.columns.swaplevel(0, 1)
data_plot_st = data_plot_st.stack('Indicator Code')
data_plot_st = data_plot_st.reset_index(drop=False)
data_plot_st.head(2)
```

Out[127]:

nom	Year	Region	Country Name	Indicator Code	val
0	1990	East Asia & Pacific	Australia	IT.NET.USER.P2	0.6
1	1990	East Asia & Pacific	Australia	NY.GDP.PCAP.CD	18249.3

In [129]:
```
# ERREMENTS QUI FONCTIONNENT - Manipulation de la dataframe : itération sur le group
internet_st = data_plot_st[data_plot_st["Indicator Code"]=='IT.NET.USER.P2']
internet_df_tab = [sub_df for name,sub_df in internet_st.groupby('Region')]
#liste des régions
li_reg = [name for name,sub_df in internet_st.groupby('Region')]
# moyenne de toutes les valeurs pour chaque région
moy_reg = np.array([df['val'].mean() for df in internet_df_tab])
# moyenne de toutes les valeurs des pays pour chaque région et chaque année
moy_reg_annees = np.array([[sub_df['val'].mean() for name,sub_df in internet_st.grou
                    for df in internet_df_tab])
moy_reg.shape, moy_reg_annees.shape
```

Out[129]: ((7,), (7, 26))

1.4.5 3.3 Graphiques

Régions, par indicateur : - tracé des violin plot pour chaque région : moyenne, médiane, écart-type
Pour une région par indicateur, pour les différents pays : - valeur classées dans l'ordre (barres) par pays pour l'année 2015, et éléments pour inférer l'évolution sur plusieurs années

In [130]:
```
def label_graphs (x_lab, y_lab, title):
    plt.xticks(rotation=25, fontsize = 12, ha='right', va='top'), plt.yticks(fontsize
    plt.xlabel(x_lab, fontsize = 16, labelpad = 20)
    plt.ylabel(y_lab, fontsize = 16, labelpad = 20)
    plt.title(title, color='k',fontsize = 18, fontweight = 'bold')
```

1. Public visé :

- Enrolment in upper secondary education (number) UIS.E.3
- Enrolment in post-secondary non-tertiary education (number) UIS.E.4
- Population dans la tranche 15-24 ans (number) SP.POP.1524.TO.UN

In [131]:
```
# Sélection des données
UIE_E_st = data_plot_st[[i in ['UIS.E.3','UIS.E.4'] for i in data_plot_st["Indicator
```

```
In [132]: SP_POP_st = data_c[data_c["Indicator Code"]=='SP.POP.1524.TO.UN']\
                      .drop(columns = ["Country Code","Indicator Name"], inplace = False)
          SP_POP_st = SP_POP_st.set_index(["Region", "Country Name", "Indicator Code"], inplace
          SP_POP_st.columns = pd.MultiIndex.from_product([SP_POP_st.columns, ['val']], names =
          SP_POP_st.columns = SP_POP_st.columns.swaplevel(0, 1)
          SP_POP_st = SP_POP_st.stack('Year')
          SP_POP_st = SP_POP_st.reset_index(drop=False)
          SP_POP_st.head(2)
```

```
Out[132]: nom      Region Country Name     Indicator Code  Year       val
          0    South Asia  Afghanistan  SP.POP.1524.TO.UN  1990 2423555.0
          1    South Asia  Afghanistan  SP.POP.1524.TO.UN  1991 2587510.0
```

```
In [133]: li = ['2010','2011','2012','2013','2014']
          data_li = UIE_E_st[[y in li for y in UIE_E_st["Year"]]]
          data_li3 = data_li[data_li["Indicator Code"]=='UIS.E.3']
          data_li4 = data_li[data_li["Indicator Code"]=='UIS.E.4']

          data_li_EAP = data_li[data_li["Region"]=='East Asia & Pacific']
          data_li_EAP3 = data_li3[data_li["Region"]=='East Asia & Pacific']
          data_li_EAP4 = data_li4[data_li["Region"]=='East Asia & Pacific']

          data_li_MENA = data_li[data_li["Region"]=='Middle East & North Africa']
          data_li_MENA = data_li[data_li["Region"]=='Middle East & North Africa']

          data_li_ECA = data_li[data_li["Region"]=='Europe & Central Asia']
          data_li_ECA4 = data_li4[data_li["Region"]=='Europe & Central Asia']

          data_li_NA = data_li[data_li["Region"]=='North America']
          data_li_SSA = data_li[data_li["Region"]=='Sub-Saharan Africa']

          data_li_SA = data_li[data_li["Region"]=='South Asia']
          data_li_SA3 = data_li3[data_li["Region"]=='South Asia']
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: UserWarning: Boolean Series
  import sys
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: Boolean Series

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: UserWarning: Boolean Serie

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:20: UserWarning: Boolean Serie
```

```
In [134]: # moyennes années 2010 à 2014 pour le secondaire, tertiaire
          moy_an_data_li_sec = data_li.groupby(['Country Name', "Indicator Code", 'Region']).me
          moy_an_data_li_sec.index = moy_an_data_li_sec.index.swaplevel(0, 1).swaplevel(1, 2)
          moy_an_data_li_sec = moy_an_data_li_sec.loc[idx['UIS.E.3',:,:]]
          moy_an_data_li_sec = moy_an_data_li_sec.sort_values('val', ascending = False, inplace
          moy_an_data_li_sec = moy_an_data_li_sec.reset_index(drop = False)
```

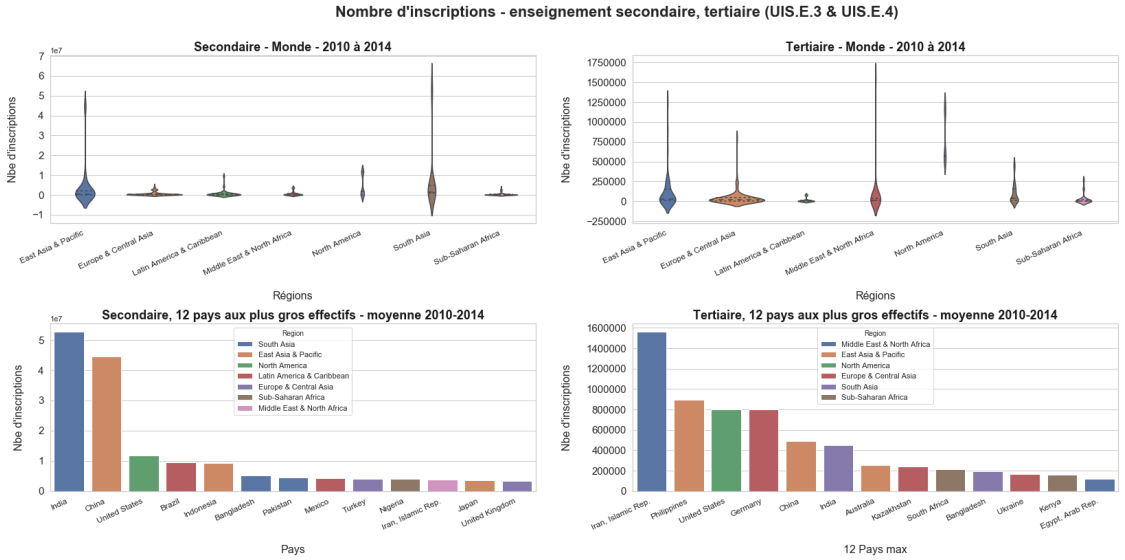
```
moy_an_data_li_ter = data_li.groupby(['Country Name', "Indicator Code", 'Region']).me
moy_an_data_li_ter.index = moy_an_data_li_ter.index.swaplevel(0, 1).swaplevel(1, 2)
moy_an_data_li_ter = moy_an_data_li_ter.loc[idx['UIS.E.4',:,:]]
moy_an_data_li_ter = moy_an_data_li_ter.sort_values('val', ascending = False, inplace
moy_an_data_li_ter = moy_an_data_li_ter.reset_index(drop = False)

moy_an_data_li_ter.head()
```

Out[134]:

nom	Region	Country Name	val
0	Middle East & North Africa	Iran, Islamic Rep.	1564102.0
1	East Asia & Pacific	Philippines	896580.0
2	North America	United States	799251.3
3	Europe & Central Asia	Germany	798918.0
4	East Asia & Pacific	China	488782.8

In [135]:
```
# Affichage

fig = plt.figure(figsize = (18, 8))
grid = plt.GridSpec(2, 4, wspace=0.5, hspace=0.6)
np.warnings.filterwarnings('ignore')

fig.suptitle("Nombre d'inscriptions - enseignement secondaire, tertiaire (UIS.E.3 & U
             x=0.02, y=1.025, ha='left', va='top', fontsize=22, fontweight = 'bold')

ax1 = plt.subplot(grid[0, 0:2])
ax1 = sns.violinplot(x="Region", y='val',# hue="Indicator Code",\
        data=data_li[data_li["Indicator Code"]=='UIS.E.3'],
                     split = False, inner="quartile", bw=.3, scale="count")
label_graphs("Régions","Nbe d'inscriptions", "Secondaire - Monde - 2010 à 2014")
#plt.legend(loc = 'upper left')

ax2 = plt.subplot(grid[0, 2:4])
ax2 = sns.violinplot(x="Region", y='val',# hue="Indicator Code",\
        data=data_li[data_li["Indicator Code"]=='UIS.E.4'],
                     split = False, inner="quartile", bw=.3, scale="count")
label_graphs("Régions","Nbe d'inscriptions", "Tertiaire - Monde - 2010 à 2014")
#plt.legend(loc = 'upper left')

ax3 = plt.subplot(grid[1, 0:2])
ax3 = sns.barplot(x="Country Name", y="val", hue = "Region",dodge = False,
                  data=moy_an_data_li_sec.iloc[:13])
label_graphs("Pays","Nbe d'inscriptions",
        "Secondaire, 12 pays aux plus gros effectifs - moyenne 2010-2014")

ax4 = plt.subplot(grid[1, 2:4])
ax4 = sns.barplot(x="Country Name", y="val", hue = "Region",dodge = False,
                  data=moy_an_data_li_ter.iloc[:13])
```

```
label_graphs("12 Pays max","Nbe d'inscriptions",
        "Tertiaire, 12 pays aux plus gros effectifs - moyenne 2010-2014")

plt.subplots_adjust(left=-0.3, bottom=-0.2, right=0.9, top=0.9, wspace=0.05, hspace=(
plt.savefig('01_public_vise.pdf')
plt.show()
```



2. Possibilités de déploiement dans le pays

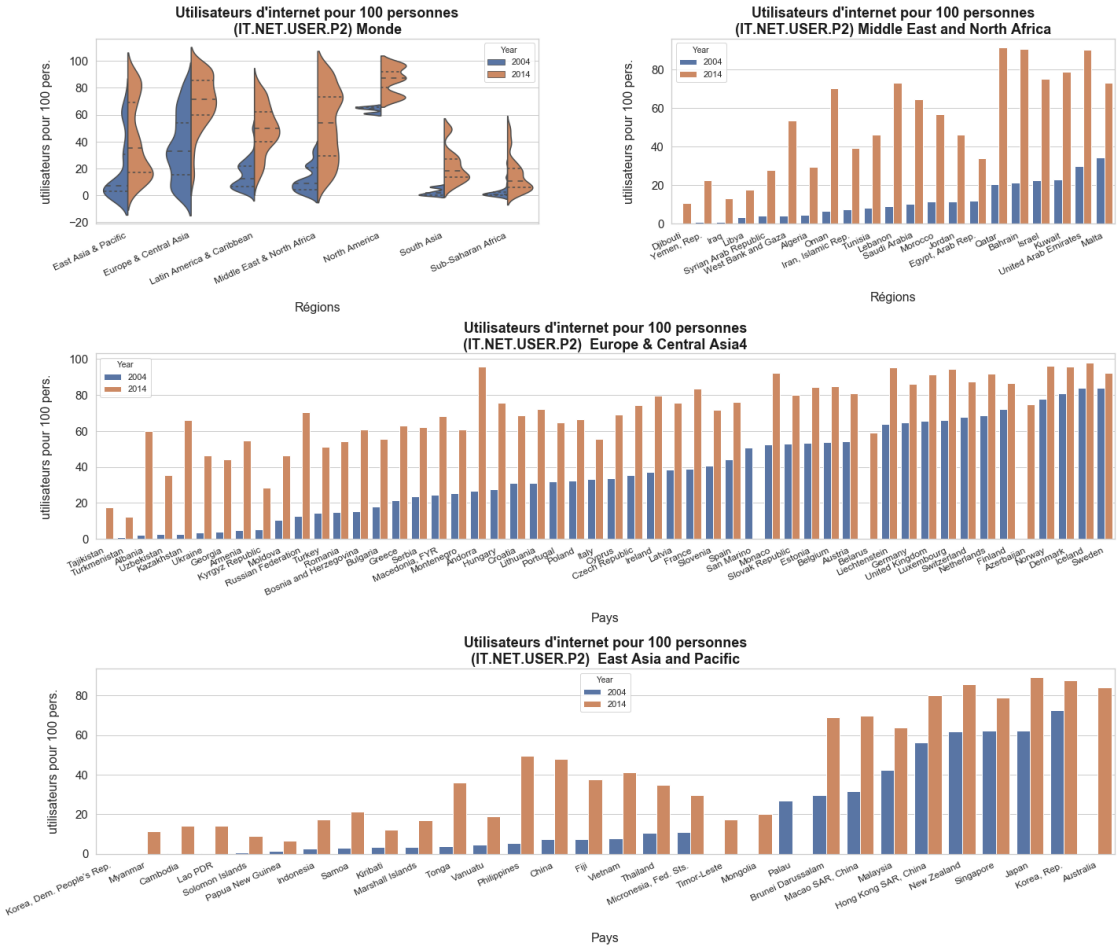
- Internet users per 100 people IT.NET.USER.P2

```
In [137]: # Sélecion des données
          internet_st = data_plot_st[data_plot_st["Indicator Code"]=='IT.NET.USER.P2']
          internet_st.head(2)
```

```
Out[137]: nom  Year              Region       Country Name  Indicator Code  val
          0    1990  East Asia & Pacific          Australia  IT.NET.USER.P2  0.6
          4    1990  East Asia & Pacific  Brunei Darussalam  IT.NET.USER.P2  0.0
```

```
In [140]: # Affichage

          fig = plt.figure(figsize = (18,18))
          grid = plt.GridSpec(3, 2, wspace=0.3, hspace=0.7)
          np.warnings.filterwarnings('ignore')

          li = ['2014','2004']
          data_li = internet_st[[y in li for y in internet_st["Year"]]]
          data_li_EAP = data_li[data_li["Region"]=='East Asia & Pacific']
          data_li_MENA = data_li[data_li["Region"]=='Middle East & North Africa']
```

```
data_li_ECA = data_li[data_li["Region"]=='Europe & Central Asia']

ax1 = plt.subplot(grid[0, 0])
ax1 = sns.violinplot(x="Region", y='val', hue="Year",\
        data=data_li, split = True, inner="quartile", bw=.3, scale="count")
label_graphs("Régions","utilisateurs pour 100 pers.",
        "Utilisateurs d'internet pour 100 personnes\n(IT.NET.USER.P2) Monde")

ax2 = plt.subplot(grid[0, 1])
ax2 = sns.barplot(x="Country Name", y="val", hue = "Year",data=data_li_MENA.sort_valu
label_graphs("Régions","utilisateurs pour 100 pers.",
        "Utilisateurs d'internet pour 100 personnes\n(IT.NET.USER.P2) Middle East and

ax3 = plt.subplot(grid[1, :])
ax3 = sns.barplot(x="Country Name", y="val", hue = "Year", data=data_li_ECA.sort_valu
label_graphs("Pays","utilisateurs pour 100 pers.",
        "Utilisateurs d'internet pour 100 personnes\n(IT.NET.USER.P2)  Europe & Centr

ax4 = plt.subplot(grid[2, :])
ax4 = sns.barplot(x="Country Name", y="val", hue = "Year", data=data_li_EAP.sort_valu
label_graphs("Pays","utilisateurs pour 100 pers.",
        "Utilisateurs d'internet pour 100 personnes\n(IT.NET.USER.P2)  East Asia and

plt.subplots_adjust(left=-0.1, bottom=0.1, right=0.9, top=0.9, wspace=0.05, hspace=0

plt.savefig('02_internet.pdf')
plt.show()
# for reg in li_reg:
#     ax1.annotate(reg, xy=(0, 0), xytext=(-17, 3),
#         textcoords='offset points', ha='left', va='bottom' )
```



3. Besoins en matière d'éducation

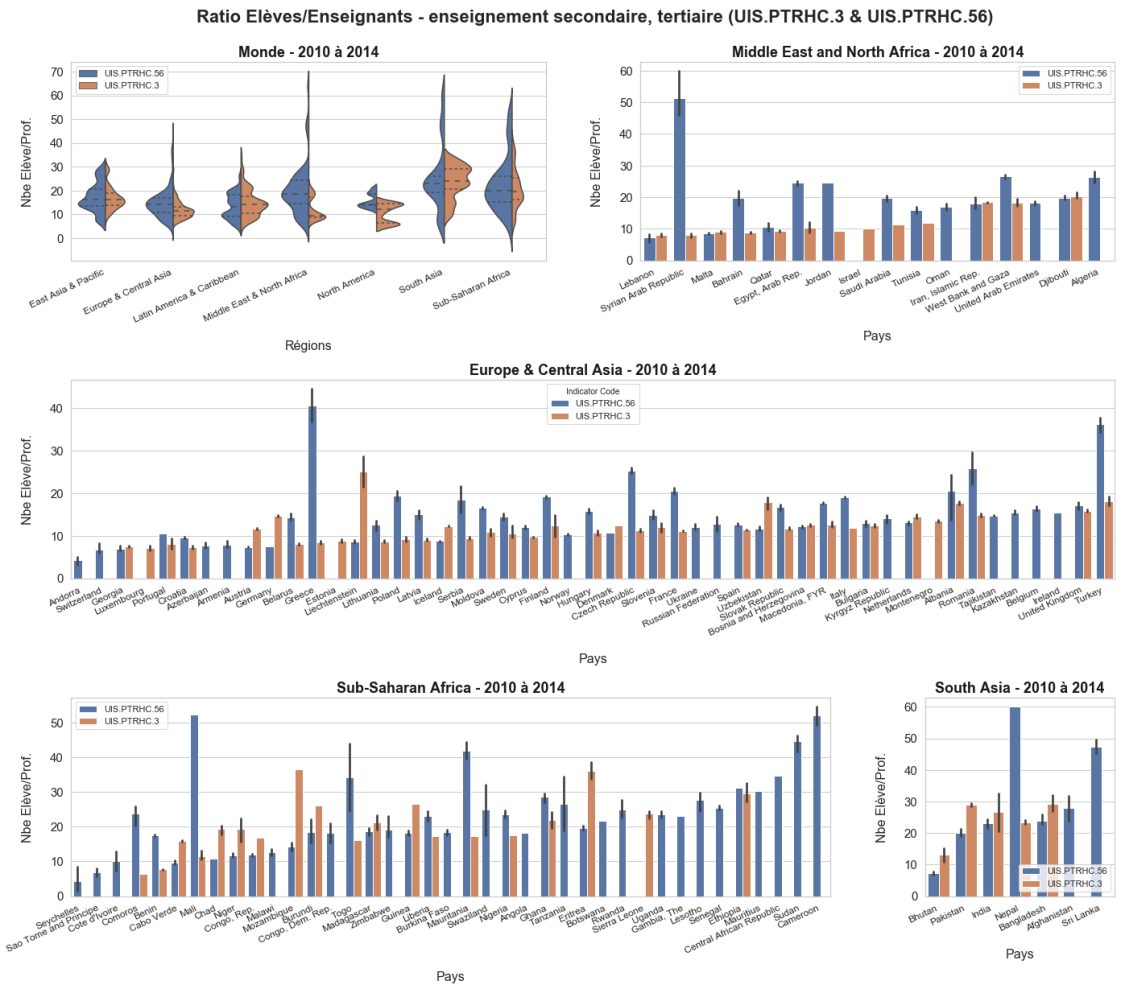
- Pupil/teacher ratio in upper secondary UIS.PTRHC.3
- Pupil/teacher ratio in tertiary UIS.PTRHC.56

```
In [141]: # Sélection des données
          PTRHC_st = data_plot_st[[i in ['UIS.PTRHC.3','UIS.PTRHC.56'] for i in data_plot_st["
          PTRHC_st.head(2)
```

```
Out[141]: nom  Year              Region Country Name Indicator Code  val
          3    1990  East Asia & Pacific    Australia   UIS.PTRHC.56 17.4
          21   1990  East Asia & Pacific    Indonesia   UIS.PTRHC.56 11.5
```

```
In [142]: # Affichage

          fig = plt.figure(figsize = (18,18))
          grid = plt.GridSpec(3, 4, wspace=0.5, hspace=0.6)
          np.warnings.filterwarnings('ignore')
```

```
li = ['2010','2011','2012','2013','2014']
data_li = PTRHC_st[[y in li for y in PTRHC_st["Year"]]]
data_li_EAP = data_li[data_li["Region"]=='East Asia & Pacific']
data_li_MENA = data_li[data_li["Region"]=='Middle East & North Africa']
data_li_ECA = data_li[data_li["Region"]=='Europe & Central Asia']
data_li_NA = data_li[data_li["Region"]=='North America']
data_li_SSA = data_li[data_li["Region"]=='Sub-Saharan Africa']
data_li_SA = data_li[data_li["Region"]=='South Asia']

fig.suptitle("Ratio Elèves/Enseignants - enseignement secondaire, tertiaire (UIS.PTR
             x=0.02, y=0.95, ha='left', va='top', fontsize=22, fontweight = 'bold')

ax1 = plt.subplot(grid[0, 0:2])
ax1 = sns.violinplot(x="Region", y='val', hue="Indicator Code",\
        data=data_li, split = True, inner="quartile", bw=.3, scale="count")
label_graphs("Régions","Nbe Elève/Prof.",
        "Monde - 2010 à 2014")
plt.legend(loc = 'upper left')

ax2 = plt.subplot(grid[0, 2:4])
ax2 = sns.barplot(x="Country Name", y="val", hue = "Indicator Code",data=data_li_MEN
label_graphs("Pays","Nbe Elève/Prof.",
        "Middle East and North Africa - 2010 à 2014")
plt.legend(loc = 'upper right')

ax3 = plt.subplot(grid[1, :])
ax3 = sns.barplot(x="Country Name", y="val", hue = "Indicator Code", data=data_li_EC
label_graphs("Pays","Nbe Elève/Prof.",
        "Europe & Central Asia - 2010 à 2014")

ax4 = plt.subplot(grid[2, 0:3])
ax4 = sns.barplot(x="Country Name", y="val", hue = "Indicator Code", data=data_li_SS
label_graphs("Pays","Nbe Elève/Prof.",
        "Sub-Saharan Africa - 2010 à 2014")
plt.legend(loc = 'upper left')

ax5 = plt.subplot(grid[2, 3])
ax5 = sns.barplot(x="Country Name", y="val", hue = "Indicator Code", data=data_li_SA
label_graphs("Pays","Nbe Elève/Prof.",
        "South Asia - 2010 à 2014")
plt.legend(loc = 'lower right')

plt.subplots_adjust(left=-0.1, bottom=0.1, right=0.9, top=0.9, wspace=0.05, hspace=0
plt.savefig('03_education.pdf')
plt.show()
```



4. Possibilités de financement

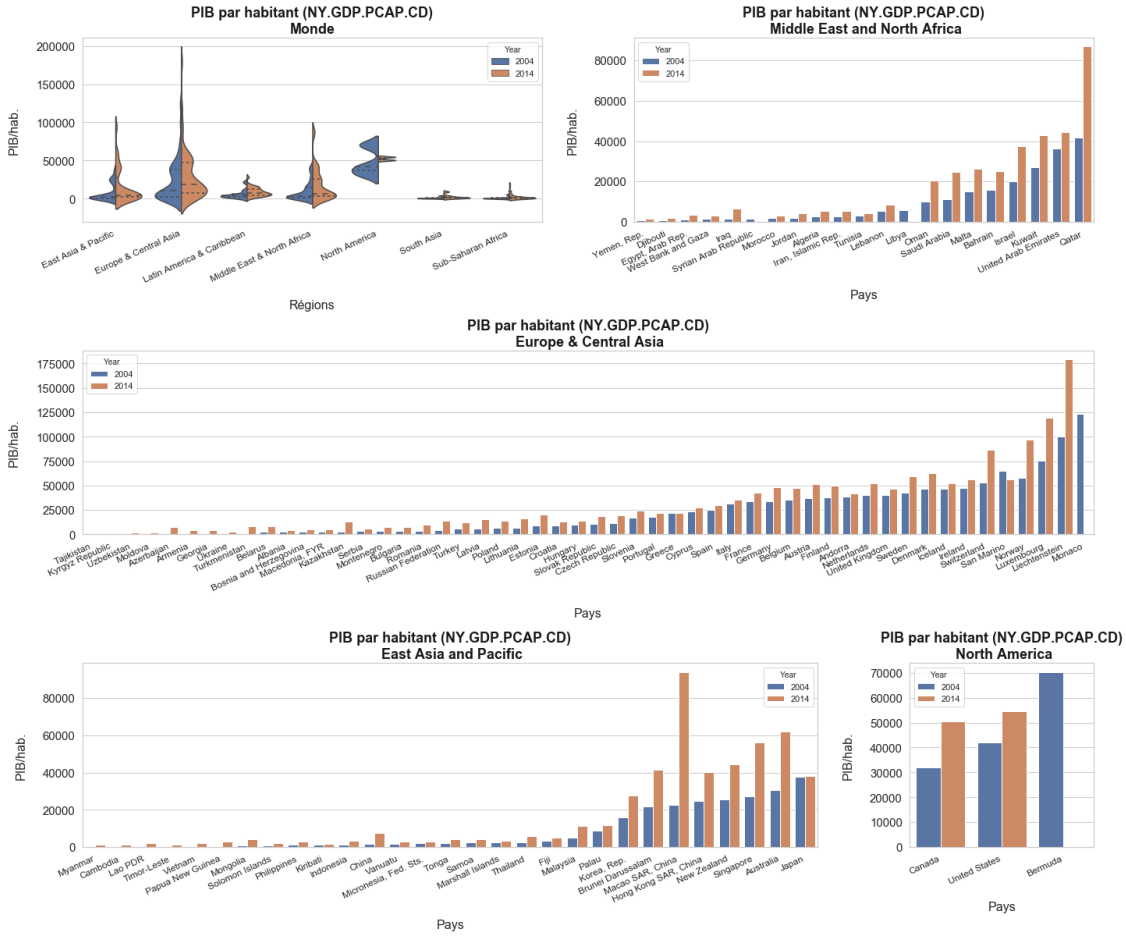
- GDP per capita (current USD) NY.GDP.PCAP.CD

```
In [143]: # Sélection des données
          GDP_st = data_plot_st[data_plot_st["Indicator Code"]=='NY.GDP.PCAP.CD']
          GDP_st.head(2)
```

```
Out[143]: nom  Year              Region       Country Name  Indicator Code     val
          1    1990  East Asia & Pacific          Australia  NY.GDP.PCAP.CD 18249.3
          5    1990  East Asia & Pacific  Brunei Darussalam  NY.GDP.PCAP.CD 13604.2
```

```
In [144]: # Affichage

          fig = plt.figure(figsize = (18,18))
          grid = plt.GridSpec(3, 4, wspace=0.5, hspace=0.7)
          np.warnings.filterwarnings('ignore')
```

```
li = ['2014','2004']
data_li = GDP_st[[y in li for y in GDP_st["Year"]]]
data_li_EAP = data_li[data_li["Region"]=='East Asia & Pacific']
data_li_MENA = data_li[data_li["Region"]=='Middle East & North Africa']
data_li_ECA = data_li[data_li["Region"]=='Europe & Central Asia']
data_li_NA = data_li[data_li["Region"]=='North America']

ax1 = plt.subplot(grid[0, 0:2])
ax1 = sns.violinplot(x="Region", y='val', hue="Year",\
        data=data_li, split = True, inner="quartile", bw=.3, scale="count")
label_graphs("Régions","PIB/hab.",
        "PIB par habitant (NY.GDP.PCAP.CD)\nMonde")

ax2 = plt.subplot(grid[0, 2:4])
ax2 = sns.barplot(x="Country Name", y="val", hue = "Year",data=data_li_MENA.sort_valu
label_graphs("Pays","PIB/hab.",
        "PIB par habitant (NY.GDP.PCAP.CD)\n Middle East and North Africa")

ax3 = plt.subplot(grid[1, :])
ax3 = sns.barplot(x="Country Name", y="val", hue = "Year", data=data_li_ECA.sort_valu
label_graphs("Pays","PIB/hab.",
        "PIB par habitant (NY.GDP.PCAP.CD)\n Europe & Central Asia")

ax4 = plt.subplot(grid[2, 0:3])
ax4 = sns.barplot(x="Country Name", y="val", hue = "Year", data=data_li_EAP.sort_valu
label_graphs("Pays","PIB/hab.",
        "PIB par habitant (NY.GDP.PCAP.CD)\n East Asia and Pacific")

ax5 = plt.subplot(grid[2, 3])
ax5 = sns.barplot(x="Country Name", y="val", hue = "Year", data=data_li_NA.sort_value
label_graphs("Pays","PIB/hab.",
        "PIB par habitant (NY.GDP.PCAP.CD)\n North America")

plt.subplots_adjust(left=-0.1, bottom=0.1, right=0.9, top=0.9, wspace=0.05, hspace=0
plt.savefig('04_economie.pdf')
plt.show()
```


PIB par habitant (NY.GDP.PCAP.CD)
Monde
PIB/hab.
Régions
East Asia & Pacific
Europe & Central Asia
Latin America & Caribbean
Middle East & North Africa
North America
South Asia
Sub-Saharan Africa
PIB par habitant (NY.GDP.PCAP.CD)
Middle East and North Africa
Pays
PIB par habitant (NY.GDP.PCAP.CD)
Europe & Central Asia
Pays
PIB par habitant (NY.GDP.PCAP.CD)
East Asia and Pacific
Pays
PIB par habitant (NY.GDP.PCAP.CD)
North America
Canada
United States
Bermuda
Pays
Year
2004
2014

In [ ]:

In [ ]: