# seance6_graphes_correction

July 2, 2023

# 1 Graphes - correction

Correction des exercices sur les graphes avec matplotlib.
Pour avoir des graphiques inclus dans le notebook, il faut ajouter cette ligne et l'exécuter en premier.

[1]:
```
%matplotlib inline
```

On change le style pour un style plus moderne, celui de ggplot :

[2]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[2]:
```
<IPython.core.display.HTML object>
```

## 1.1 Données

### 1.1.1 élections

Pour tous les exemples qui suivent, on utilise les résultat élection présidentielle de 2012. Si vous n'avez pas le module actuariat_python, il vous suffit de recopier le code de la fonction elections_presidentielles qui utilise la fonction read_excel :

[3]:
```
from actuariat_python.data import elections_presidentielles
dict_df = elections_presidentielles(local=True, agg="dep")
```

[4]:
```
def cleandep(s):
    if isinstance(s, str):
        r = s.lstrip('0')
    else:
        r = str(s)
    return r
dict_df["dep1"]["Code du département"] = dict_df["dep1"]["Code du département"].
 ↪apply(cleandep)
dict_df["dep2"]["Code du département"] = dict_df["dep2"]["Code du département"].
 ↪apply(cleandep)
```

[5]:
```
deps = dict_df["dep1"].merge(dict_df["dep2"],
                                        on="Code du département",
                                        suffixes=("T1", "T2"))
deps["rHollandeT1"] = deps['François HOLLANDE (PS)T1'] / (deps["VotantsT1"] -␣
 ↪deps["Blancs et nulsT1"])
deps["rSarkozyT1"] = deps['Nicolas SARKOZY (UMP)T1'] / (deps["VotantsT1"] -␣
 ↪deps["Blancs et nulsT1"])
```

```
deps["rNulT1"] = deps["Blancs et nulsT1"] / deps["VotantsT1"]
deps["rHollandeT2"] = deps["François HOLLANDE (PS)T2"] / (deps["VotantsT2"] -␣
↪deps["Blancs et nulsT2"])
deps["rSarkozyT2"] = deps['Nicolas SARKOZY (UMP)T2'] / (deps["VotantsT2"] -␣
↪deps["Blancs et nulsT2"])
deps["rNulT2"] = deps["Blancs et nulsT2"] / deps["VotantsT2"]
data = deps[["Code du département", "Libellé du départementT1",
             "VotantsT1", "rHollandeT1", "rSarkozyT1", "rNulT1",
             "VotantsT2", "rHollandeT2", "rSarkozyT2", "rNulT2"]]
data_elections = data # parfois data est remplacé dans la suite
data.head()
```

[5]:

```
  Code du département Libellé du départementT1  VotantsT1  rHollandeT1  \
0                   1                      AIN     327812     0.227459
1                   2                    AISNE     303140     0.271027
2                   3                   ALLIER     211009     0.296824
3                   4  ALPES-DE-HAUTE-PROVENCE     102899     0.243591
4                   5             HAUTES-ALPES      88619     0.244858

   rSarkozyT1    rNulT1  VotantsT2  rHollandeT2  rSarkozyT2    rNulT2
0    0.304090  0.019685     326587     0.427692    0.572308  0.059748
1    0.241958  0.017141     302076     0.524020    0.475980  0.069704
2    0.240238  0.023975     211132     0.568861    0.431139  0.070686
3    0.254673  0.020515     103581     0.510594    0.489406  0.064095
4    0.261071  0.020786      89405     0.508935    0.491065  0.067390
```

### 1.1.2 localisation des villes

[6]:

```
from pyensae.datasource import download_data
download_data("villes_france.csv", url="http://sql.sh/ressources/sql-villes-france/")
cols = ["ncommune", "numero_dep", "slug", "nom", "nom_simple", "nom_reel",␣
↪"nom_soundex", "nom_metaphone", "code_postal",
    "numero_commune", "code_commune", "arrondissement", "canton", "pop2010",␣
↪"pop1999", "pop2012",
    "densite2010", "surface", "superficie", "dlong", "dlat", "glong", "glat", "slong",␣
↪"slat", "alt_min", "alt_max"]
import pandas
villes = pandas.read_csv("villes_france.csv", header=None,low_memory=False, names=cols)
```

## 1.2 exercice 1 : centrer la carte de la France

On recentre la carte. Seule modification : `[-5, 10, 38, 52]`.

[7]:

```
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature

fig = plt.figure(figsize=(6,6))
ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree())
ax.set_extent([-5, 10, 38, 52])

ax.add_feature(cfeature.OCEAN.with_scale('50m'))
ax.add_feature(cfeature.RIVERS.with_scale('50m'))
```

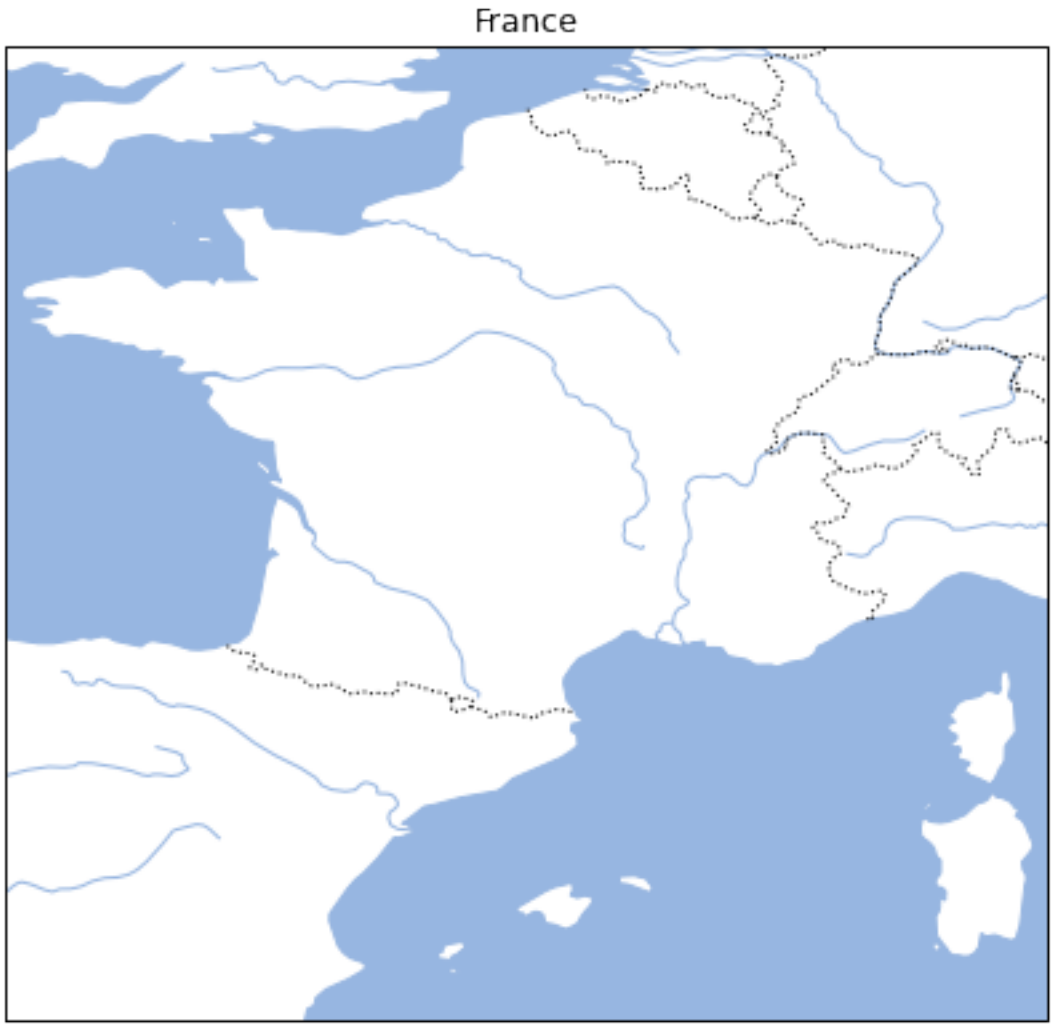
```
ax.add_feature(cfeature.BORDERS.with_scale('50m'), linestyle=':')
ax.set_title('France');
```

```
c:\python372_x64\lib\site-packages\cartopy\mpl\geoaxes.py:388:
MatplotlibDeprecationWarning:
The 'inframe' parameter of draw() was deprecated in Matplotlib 3.3 and will be
removed two minor releases later. Use Axes.redraw_in_frame() instead. If any
parameter follows 'inframe', they should be passed as keyword, not positionally.
  inframe=inframe)
c:\python372_x64\lib\site-packages\cartopy\io\__init__.py:260: DownloadWarning:
Downloading: http://naciscdn.org/naturalearth/50m/physical/ne_50m_ocean.zip
  warnings.warn('Downloading: {}'.format(url), DownloadWarning)
c:\python372_x64\lib\site-packages\cartopy\io\__init__.py:260: DownloadWarning:
Downloading:
http://naciscdn.org/naturalearth/50m/physical/ne_50m_rivers_lake_centerlines.zip
  warnings.warn('Downloading: {}'.format(url), DownloadWarning)
c:\python372_x64\lib\site-packages\cartopy\io\__init__.py:260: DownloadWarning:
Downloading: http://naciscdn.org/naturalearth/50m/cultural/ne_50m_admin_0_bounda
ry_lines_land.zip
  warnings.warn('Downloading: {}'.format(url), DownloadWarning)
```

### 1.3 exercice 2 : placer les plus grandes villes de France sur la carte

On reprend la fonction `carte_france` donnée par l'énoncé et modifié avec le résultat de la question précédente.

[8]:
```
def carte_france(figsize=(7, 7)):
    fig = plt.figure(figsize=figsize)
    ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree())
    ax.set_extent([-5, 10, 38, 52])

    ax.add_feature(cfeature.OCEAN.with_scale('50m'))
    ax.add_feature(cfeature.RIVERS.with_scale('50m'))
    ax.add_feature(cfeature.BORDERS.with_scale('50m'), linestyle=':')
    ax.set_title('France');
    return ax

carte_france();
```



On ne retient que les villes de plus de 100.000 habitants. Toutes les villes ne font pas partie de la métropole :

[9]:
```
grosses_villes = villes[villes.pop2012 > 100000][["dlong","dlat","nom", "pop2012"]]
grosses_villes.describe()
```

[9]:

| | dlong | dlat | pop2012 |
|---|---|---|---|
| count | 37.000000 | 37.000000 | 3.700000e+01 |
| mean | 4.182151 | 45.579735 | 2.432647e+05 |
| std | 7.746851 | 7.749732 | 3.434288e+05 |
| min | -4.483330 | 1.718190 | 1.035520e+05 |
| 25% | 1.083330 | 44.833300 | 1.241520e+05 |
| 50% | 3.083330 | 47.316700 | 1.496490e+05 |
| 75% | 5.433330 | 48.683300 | 2.061940e+05 |
| max | 46.710700 | 50.633300 | 2.125851e+06 |

[10]:
```
grosses_villes.sort_values("dlat").head()
```

[10]:

| | dlong | dlat | nom | pop2012 |
|---|---|---|---|---|
| 36666 | 46.71070 | 1.71819 | SAINT-DENIS | 131649 |
| 27143 | 2.88333 | 42.68330 | PERPIGNAN | 105096 |
| 33676 | 5.93333 | 43.11670 | TOULON | 160712 |
| 4439 | 5.37639 | 43.29670 | MARSEILLE | 797491 |
| 4420 | 5.43333 | 43.53330 | AIX-EN-PROVENCE | 134324 |

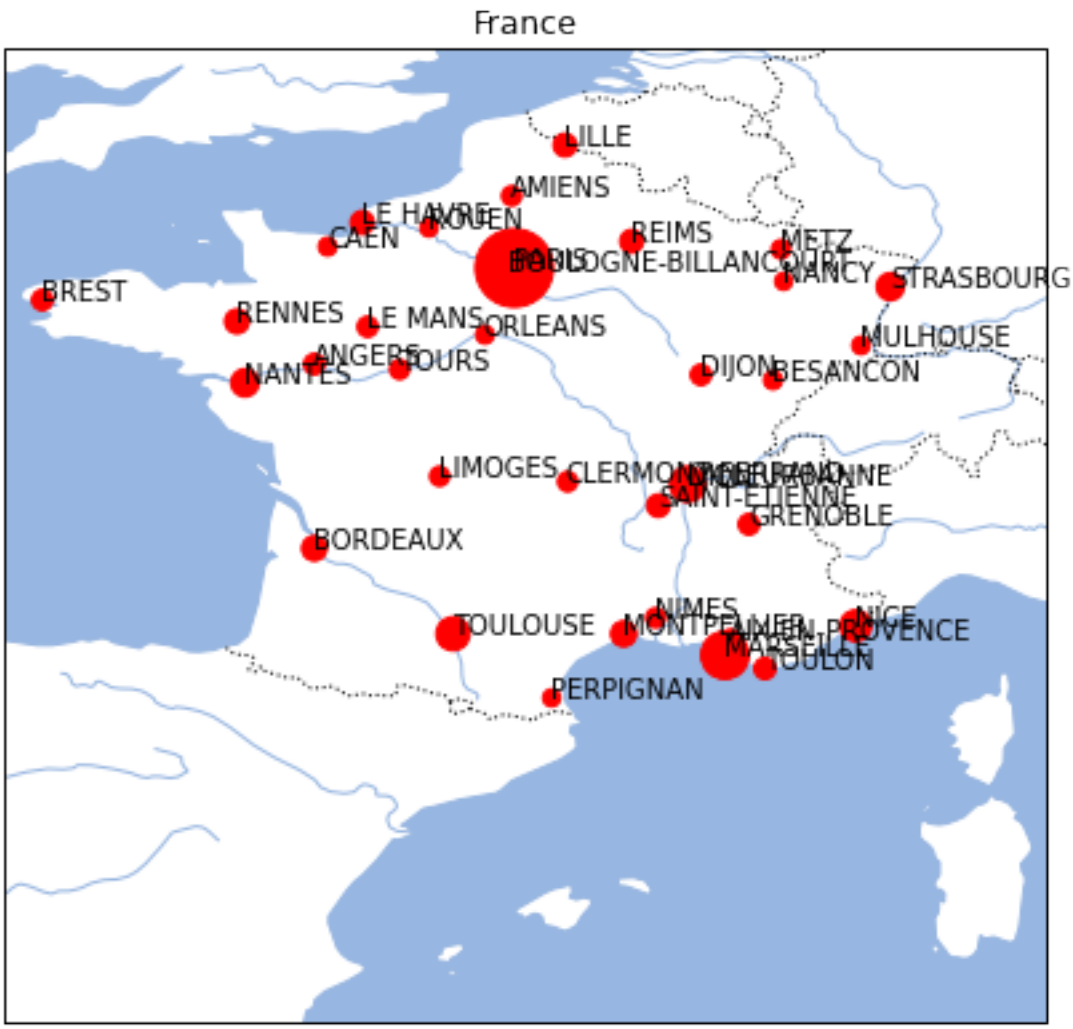
Saint-Denis est à la Réunion. On l'enlève de l'ensemble :

[11]:
```
grosses_villes = villes[(villes.pop2012 > 100000) & (villes.dlat > 40)] \
        [["dlong","dlat","nom", "pop2012"]]
```

On dessine la carte souhaitée en ajoutant un marqueur pour chaque ville dont la surface dépend du nombre d'habitant. Sa taille doit être proportionnelle à à la racine carrée du nombre d'habitants.

[12]:
```
import matplotlib.pyplot as plt
ax = carte_france()

def affiche_ville(ax, x, y, nom, pop):
    ax.plot(x, y, 'ro', markersize=pop**0.5/50)
    ax.text(x, y, nom)

for lon, lat, nom, pop in zip(grosses_villes["dlong"],
                         grosses_villes["dlat"],
                         grosses_villes["nom"],
                        grosses_villes["pop2012"]):
    affiche_ville(ax, lon, lat, nom, pop)

ax;
```

```
c:\python372_x64\lib\site-packages\cartopy\mpl\geoaxes.py:388:
MatplotlibDeprecationWarning:
The 'inframe' parameter of draw() was deprecated in Matplotlib 3.3 and will be
removed two minor releases later. Use Axes.redraw_in_frame() instead. If any
parameter follows 'inframe', they should be passed as keyword, not positionally.
  inframe=inframe)
```




**rappel : fonction zip**
La fonction zip *colle* deux séquences ensemble.

[13]:
```
list(zip([1,2,3], ["a", "b", "c"]))
```

[13]:
```
[(1, 'a'), (2, 'b'), (3, 'c')]
```

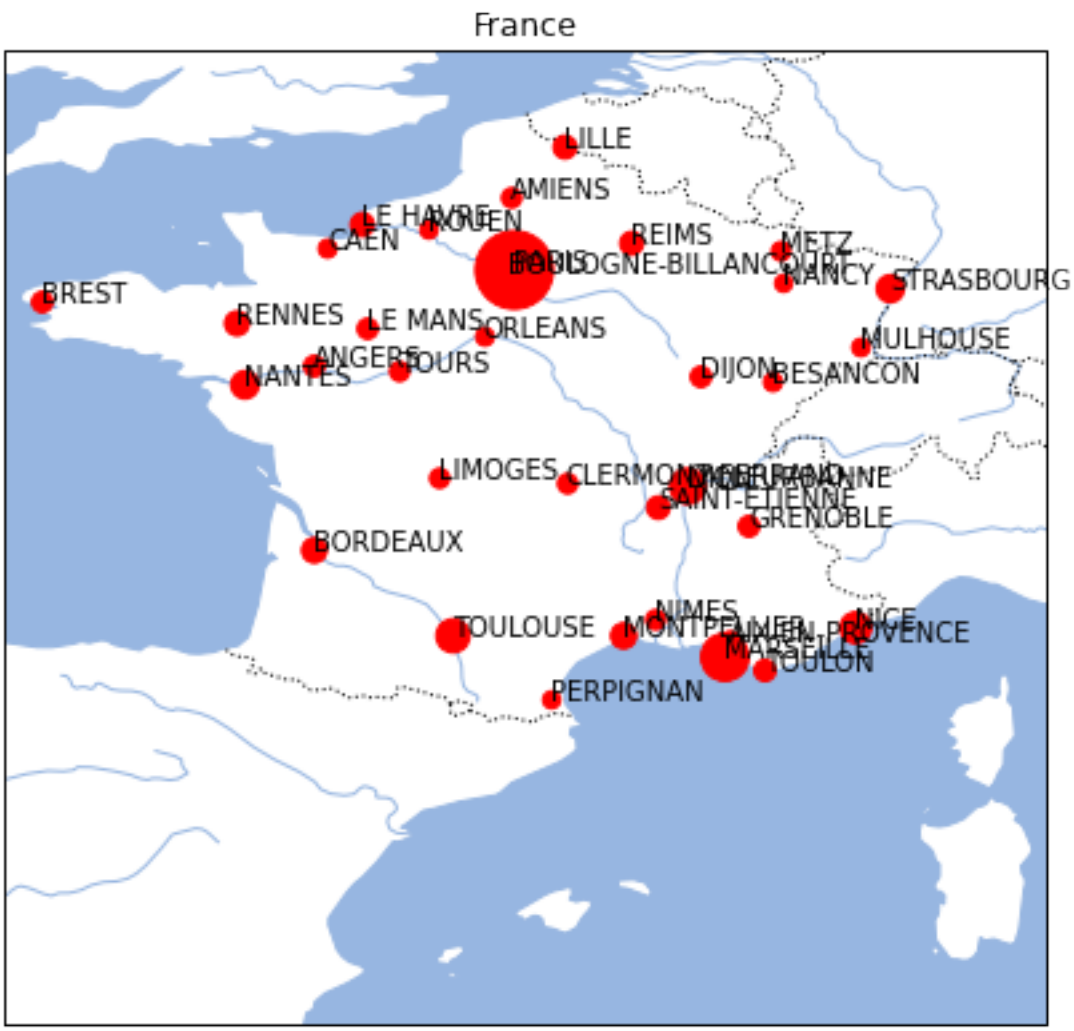
On l'utilise souvent de cette manière :

[14]:
```
for a,b in zip([1,2,3], ["a", "b", "c"]):
    # faire quelque chose avec a et b
    print(a,b)
```

```
1 a
2 b
3 c
```

Sans la fonction zip :

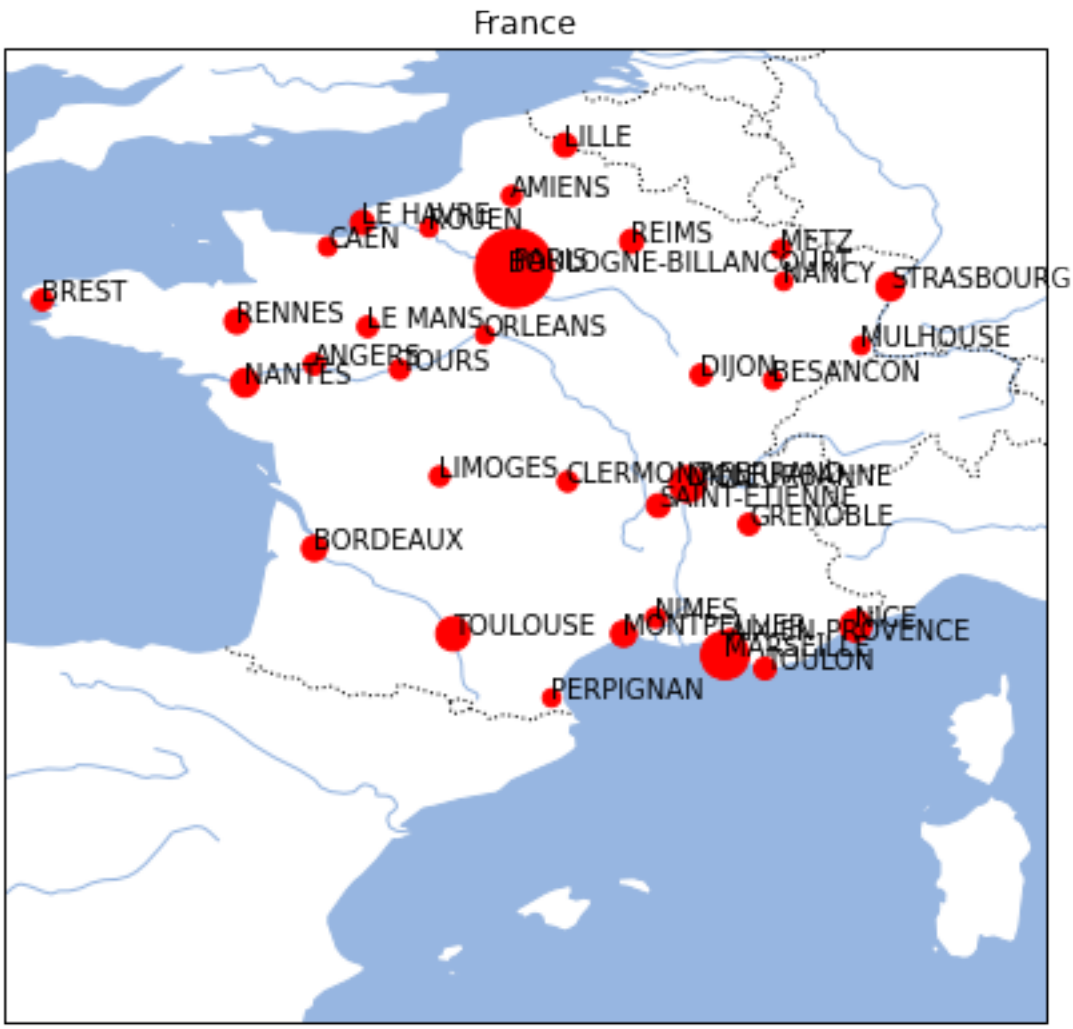
[15]:
```
ax = carte_france()

def affiche_ville(ax, x, y, nom, pop):
    ax.plot(x, y, 'ro', markersize=pop**0.5/50)
    ax.text(x, y, nom)

def affiche_row(ax, row):
    affiche_ville(ax, row["dlong"], row["dlat"], row["nom"], row["pop2012"])

grosses_villes.apply(lambda row: affiche_row(ax, row), axis=1)
ax;
```




Ou encore :

[16]:
```
import matplotlib.pyplot as plt
ax = carte_france()

def affiche_ville(ax, x, y, nom, pop):
```

```
    ax.plot(x, y, 'ro', markersize=pop**0.5/50)
    ax.text(x, y, nom)

for i in range(0, grosses_villes.shape[0]):
    ind = grosses_villes.index[i]
    # important ici, les lignes sont indexées par rapport à l'index de départ
    # comme les lignes ont été filtrées pour ne garder que les grosses villes,
    # il faut soit utiliser reset_index soit récupérer l'indice de la ligne
    lon, lat = grosses_villes.loc[ind, "dlong"], grosses_villes.loc[ind, "dlat"]
    nom, pop = grosses_villes.loc[ind, "nom"], grosses_villes.loc[ind, "pop2012"]
    affiche_ville(ax, lon, lat, nom, pop)
ax;
```




## 1.4 exercice 3 : résultats des élections par département

On reprend le résultat des élections, on construit d'abord un dictionnaire dans lequel `{ departement: vainqueur }`.

[17]:
```
data_elections.shape, data_elections[data_elections.rHollandeT2 > data_elections.
↪rSarkozyT2].shape
```

[17]:
```
((107, 10), (63, 10))
```

Il y a 63 départements où Hollande est vainqueur.

[18]:
```
hollande_gagnant = dict(zip(data_elections["Libellé du départementT1"], data_elections.
↪rHollandeT2 > data_elections.rSarkozyT2))
list(hollande_gagnant.items())[:5]
```

[18]:
```
[('AIN', False),
 ('AISNE', True),
 ('ALLIER', True),
 ('ALPES-DE-HAUTE-PROVENCE', True),
 ('HAUTES-ALPES', True)]
```

On récupère les formes de chaque département :

[19]:
```
from pyensae.datasource import download_data
try:
    download_data("GEOFLA_2-1_DEPARTEMENT_SHP_LAMB93_FXX_2015-12-01.7z",
                  website="https://wxs-telechargement.ign.fr/oikr5jryiph0iwhw36053ptm/
↪telechargement/inspire/" + \
                ␣
↪"GEOFLA_THEME-DEPARTEMENTS_2015_2$GEOFLA_2-1_DEPARTEMENT_SHP_LAMB93_FXX_2015-12-01/
↪file/")
except Exception as e:
    # au cas le site n'est pas accessible
    download_data("GEOFLA_2-1_DEPARTEMENT_SHP_LAMB93_FXX_2015-12-01.7z", website="xd")
```

[20]:
```
from pyquickhelper.filehelper import un7zip_files
try:
    un7zip_files("GEOFLA_2-1_DEPARTEMENT_SHP_LAMB93_FXX_2015-12-01.7z",␣
↪where_to="shapefiles")
    departements = 'shapefiles/GEOFLA_2-1_DEPARTEMENT_SHP_LAMB93_FXX_2015-12-01/GEOFLA/
↪1_DONNEES_LIVRAISON_2015/' + \
                    'GEOFLA_2-1_SHP_LAMB93_FR-ED152/DEPARTEMENT/DEPARTEMENT.shp'
except FileNotFoundError as e:
    # Il est possible que cette instruction ne fonctionne pas.
    # Dans ce cas, on prendra une copie de ce fichier.
    import warnings
    warnings.warn("Plan B parce que " + str(e))
    download_data("DEPARTEMENT.zip")
    departements = "DEPARTEMENT.shp"

import os
if not os.path.exists(departements):
    raise FileNotFoundError("Impossible de trouver '{0}'".format(departements))
```

[21]:
```
import shapefile
shp = departements
r = shapefile.Reader(shp)
shapes = r.shapes()
records = r.records()
```

```
records[0]
```

[21]:
```
Record #0: ['DEPARTEM0000000000000004', '89', 'YONNE', '024', 'AUXERRE', 742447,
6744261, 748211, 6750855, '27', 'BOURGOGNE-FRANCHE-COMTE']
```

Le problème est que les codes sont difficiles à associer aux résultats des élections. La page Wikipedia de Bas-Rhin lui associe le code 67. Le Bas-Rhin est orthographié `BAS RHIN` dans la liste des résultats. Le code du département n'apparaît pas dans les *shapefile* récupérés. Il faut *matcher* sur le nom du département. On met tout en minuscules et on enlève espaces et tirets.

[22]:
```
hollande_gagnant_clean = { k.lower().replace("-", "").replace(" ", ""): v for k,v in␣
↪hollande_gagnant.items()}
list(hollande_gagnant_clean.items())[:5]
```

[22]:
```
[('ain', False),
 ('aisne', True),
 ('allier', True),
 ('alpesdehauteprovence', True),
 ('hautesalpes', True)]
```

Et comme il faut aussi remplacer les accents, on s'inspire de la fonction remove_diacritic :

[23]:
```
import unicodedata

def retourne_vainqueur(nom_dep):
    s = nom_dep.lower().replace("-", "").replace(" ", "")
    nkfd_form = unicodedata.normalize('NFKD', s)
    only_ascii = nkfd_form.encode('ASCII', 'ignore')
    s = only_ascii.decode("utf8")
    if s in hollande_gagnant_clean:
        return hollande_gagnant_clean[s]
    else:
        keys = list(sorted(hollande_gagnant_clean.keys()))
        keys = [_ for _ in keys if _[0].lower() == s[0].lower()]
        print("impossible de savoir pour ", nom_dep, "*", s, "*", " --- ", keys[:5])
        return None
```

[24]:
```
import math

def lambert932WGPS(lambertE, lambertN):

    class constantes:
        GRS80E = 0.081819191042816
        LONG_0 = 3
        XS = 700000
        YS = 12655612.0499
        n = 0.7256077650532670
        C = 11754255.4261

    delX = lambertE - constantes.XS
    delY = lambertN - constantes.YS
    gamma = math.atan(-delX / delY)
    R = math.sqrt(delX * delX + delY * delY)
    latiso = math.log(constantes.C / R) / constantes.n
```

```
    sinPhiit0 = math.tanh(latiso + constantes.GRS80E * math.atanh(constantes.GRS80E *␣
↪math.sin(1)))
    sinPhiit1 = math.tanh(latiso + constantes.GRS80E * math.atanh(constantes.GRS80E *␣
↪sinPhiit0))
    sinPhiit2 = math.tanh(latiso + constantes.GRS80E * math.atanh(constantes.GRS80E *␣
↪sinPhiit1))
    sinPhiit3 = math.tanh(latiso + constantes.GRS80E * math.atanh(constantes.GRS80E *␣
↪sinPhiit2))
    sinPhiit4 = math.tanh(latiso + constantes.GRS80E * math.atanh(constantes.GRS80E *␣
↪sinPhiit3))
    sinPhiit5 = math.tanh(latiso + constantes.GRS80E * math.atanh(constantes.GRS80E *␣
↪sinPhiit4))
    sinPhiit6 = math.tanh(latiso + constantes.GRS80E * math.atanh(constantes.GRS80E *␣
↪sinPhiit5))

    longRad = math.asin(sinPhiit6)
    latRad = gamma / constantes.n + constantes.LONG_0 / 180 * math.pi

    longitude = latRad / math.pi * 180
    latitude = longRad / math.pi * 180

    return longitude, latitude

lambert932WGPS(99217.1, 6049646.300000001), lambert932WGPS(1242417.2, 7110480.
↪100000001)
```

[24]:
```
((-4.1615802638173065, 41.303505287589545),
 (10.699505053975292, 50.85243395553585))
```

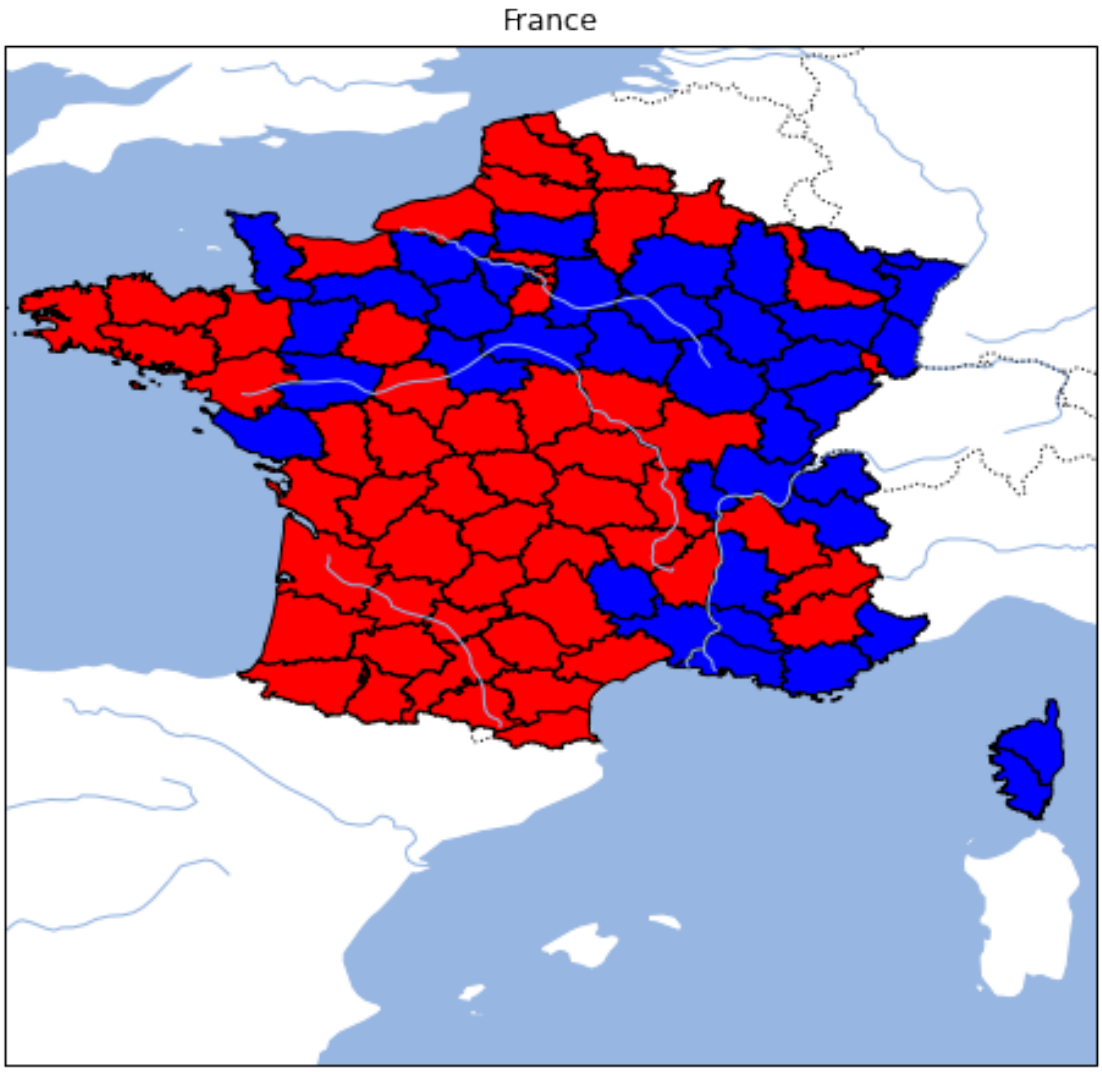
Puis on utilise le code de l'énoncé en changeant la couleur. Pas de couleur indique les départements pour lesquels on ne sait pas.

[25]:
```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
ax = carte_france((8,8))

from matplotlib.collections import LineCollection
import shapefile
import geopandas
from shapely.geometry import Polygon
from shapely.ops import cascaded_union, unary_union

shp = departements
r = shapefile.Reader(shp)
shapes = r.shapes()
records = r.records()

polys = []
colors = []
for i, (record, shape) in enumerate(zip(records, shapes)):
    # Vainqueur
    dep = retourne_vainqueur(record[2])
    if dep is not None:
        couleur = "red" if dep else "blue"
```

```
    else:
        couleur = "gray"
    # les coordonnées sont en Lambert 93
    if i == 0:
        print(record, shape.parts, couleur)
    geo_points = [lambert932WGPS(x,y) for x, y in shape.points]
    if len(shape.parts) == 1:
        # Un seul polygone
        poly = Polygon(geo_points)
    else:
        # Il faut les fusionner.
        ind = list(shape.parts) + [len(shape.points)]
        pols = [Polygon(geo_points[ind[i]:ind[i+1]]) for i in range(0, len(shape.
 ↪parts))]
        try:
            poly = unary_union(pols)
        except Exception as e:
            print("Cannot merge: ", record)
            print([_.length for _ in pols], ind)
            poly = Polygon(geo_points)
    polys.append(poly)
    colors.append(couleur)

data = geopandas.GeoDataFrame(dict(geometry=polys, colors=colors))
geopandas.plotting.plot_polygon_collection(ax, data['geometry'],␣
 ↪facecolor=data['colors'],
                                            values=None, edgecolor='black');
```

```
Record #0: ['DEPARTEM0000000000000004', '89', 'YONNE', '024', 'AUXERRE', 742447,
6744261, 748211, 6750855, '27', 'BOURGOGNE-FRANCHE-COMTE'] [0] blue
```

```
c:\python372_x64\lib\site-packages\cartopy\mpl\geoaxes.py:388:
MatplotlibDeprecationWarning:
The 'inframe' parameter of draw() was deprecated in Matplotlib 3.3 and will be
removed two minor releases later. Use Axes.redraw_in_frame() instead. If any
parameter follows 'inframe', they should be passed as keyword, not positionally.
  inframe=inframe)
```




La fonction fait encore une erreur pour la Corse du Sud… Je la laisse en guise d'exemple.

### 1.5 exercice 3 avec les shapefile etalab

Les données sont disponibles sur GEOFLA® Départements mais vous pouvez reprendre le code ce-dessus pour les télécharger.

[26]:
```
# ici, il faut dézipper manuellement les données
# à terminer
```

### 1.6 exercice 4 : même code, widget différent

On utilise des checkbox pour activer ou désactiver l'un des deux candidats.

[27]:
```
import matplotlib.pyplot as plt
from ipywidgets import interact, Checkbox
```

```
def plot(candh, cands):

    fig, axes = plt.subplots(1, 1, figsize=(14,5), sharey=True)
    if candh:
        data_elections.plot(x="rHollandeT1", y="rHollandeT2", kind="scatter",␣
 ↪label="H", ax=axes)
    if cands:
        data_elections.plot(x="rSarkozyT1", y="rSarkozyT2", kind="scatter", label="S",␣
 ↪ax=axes, c="red")
    axes.plot([0.2,0.7], [0.2,0.7], "g--")
    return axes

candh = Checkbox(description='Hollande', value=True)
cands = Checkbox(description='Sarkozy', value=True)

interact(plot, candh=candh, cands=cands);
```

```
interactive(children=(Checkbox(value=True, description='Hollande'), Checkbox(value=True,␣
 ↪description='Sarkozy'...
```

Si aucune case à cocher n'apparaît, il faut se reporter à l'installation du module ipywidgets. La cause la plus probable est l'oubli la seconde ligne

```
pip install ipywidgets
jupyter nbextension enable --py widgetsnbextension
```

qui active l'extension. Il faut relancer le serveur de notebook après l'avoir exécutée depuis la ligne de commande.

[28] :

```

```

[29] :

```

```