election_carte_electorale_correction

July 2, 2023

1 Elections et cartes électorales - correction

Bidouiller les cartes électorales n'est pas facile mais il n'est pas nécessaire d'être très efficace pour orienter la décision dans un sens ou dans l'autre. L'idée principale consiste à bouger des électeurs d'une circonscription à l'autre pour favoriser les candidats d'un seul parti. Il faut que ces candidats sont élus avec une majorité suffisante tandis que les candidats adversaires doivent l'être avec une grande majorité. C'est une façon de donner plus d'importance aux voix d'un seul parti car elles annulent celles des autres. L'objectif visé est la préparation d'une prochaine élection à partir des résultats de la précédente sans que cela se voit trop. Mais nous pourrions essayer de faire basculer les résultats d'une élection dans un camp ou dans l'autre.

[1]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[1]:
```
<IPython.core.display.HTML object>
```

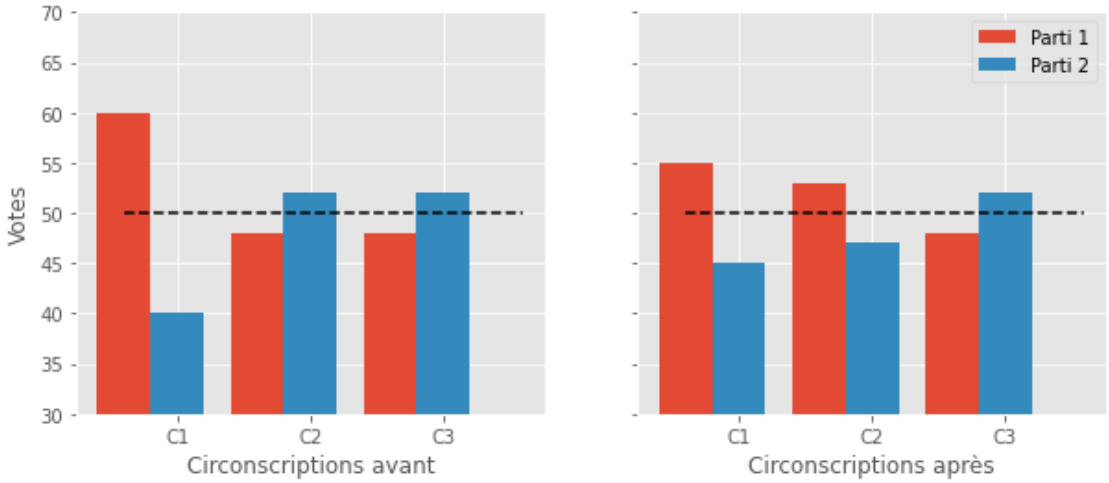
[2]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

1.1 Plan

Voici un exemple avec trois circonscriptions voisines et de taille équivalente où le candidat non majoritaire sur les trois circonscriptions a gagné largement sa circonscription. Il pourrait peut-être utiliser certaines des voix au-dessus des 50% pour être moins distancé sur une autre circonscription.

[3]:
```
fig, ax = plt.subplots(1, 2, sharey=True, figsize=(10,4))
ind = [1, 2, 3]
ind2 = [_ + 0.4 for _ in ind]
P1 = (60, 48, 48)
P2 = (40, 52, 52)
ax[0].bar(ind, P1, label="Parti 1", width=0.4)
ax[0].bar(ind2, P2, label="Parti 2", width=0.4)
ax[0].plot([1,4], [50, 50], "--", color="black")
ax[0].set_xlabel('Circonscriptions avant')
plt.setp(ax, xticks=ind2, xticklabels=('C1', 'C2', 'C3'))
P1 = (55, 53, 48)
P2 = (45, 47, 52)
ax[1].bar(ind, P1, label="Parti 1", width=0.4)
ax[1].bar(ind2, P2, label="Parti 2", width=0.4)
ax[1].plot([1,4], [50, 50], "--", color="black")
ax[1].set_xlabel('Circonscriptions après')
ax[0].set_ylabel('Votes')
```

```
plt.ylim([30, 70])
plt.legend();
```



les moyens

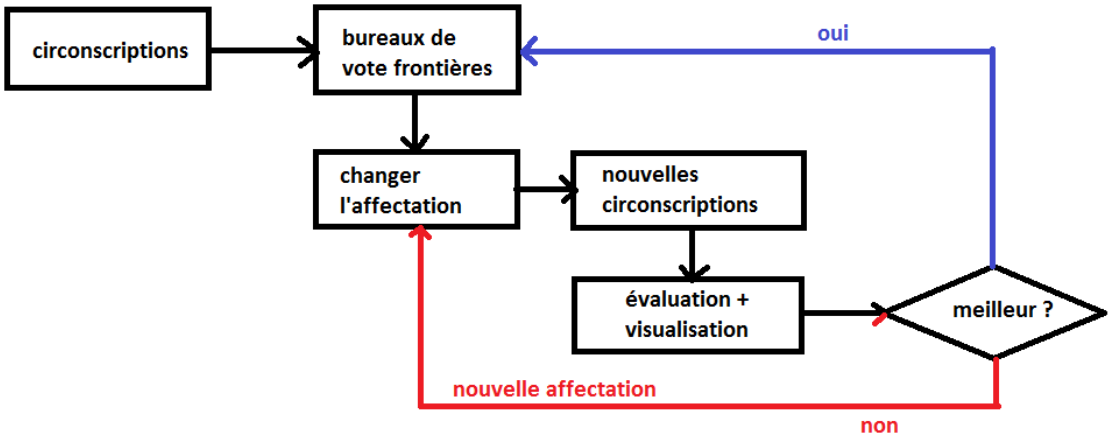
Nous ne connaissons pas les votes de chaque électeurs mais nous connaissons les résultats agrégés au niveau des bureaux de vote. Nous ne pouvons pas influencer les résultats de l'élection présidentielle car les votes sont agrégées au niveau du pays : une voix à Perpignan compte autant d'une voix à Charleville-Mézières. C'est différent pour les élections législatives. Un vote à Charleville n'a qu'un impact dans l'une des 577 circonscriptions. Que se passe-t-il alors si on fait basculer un bureau de vote d'une circonscription à une autre ?

la stratégie

Travailler à plusieurs nécessite de répartir de travailler et d'isoler quelques fonctionnalités qui peuvent être développées en parallèle. Le premier besoin essentiel est celui de la **visualisation** des résultats. Nous allons faire beaucoup d'essais et il faut pouvoir rapidement visualiser le résultat afin d'éviter les erreurs visuellement évidentes. Comme tout projet, il faut un critère numérique qui permette de résumer la qualité d'une solution. Dans notre cas, celui-ci est relié aux nombres de députés élus venant du parti à favoriser. Le second besoin est l'**évaluation** d'une solution. Qu'est ce qui définit une solution ? Ce sont la description des circonscriptions, autrement l'appariement des bureaux de votes aux circonscriptions. Il faut réfléchir à un mécanisme qui nous permette de générer plusieurs solutions, plusieurs appariements. C'est l'étape de **génération** des solutions. C'est sans doute cette dernière partie qui sera la plus complexe car elle doit s'intéresser aux voisinages de bureaux de votes. On peut isoler un traitement spécifique qui consiste à calculer les **voisins** à regarder à partir d'une définition des circonscriptions.

[4]:
```
from pyquickhelper.helpgen import NbImage
NbImage("gerrysol.png")
```

[4]:



travailler en commun
Si chaque tâche, visualisation, évaluation, génération, peut être conçues en parallèle, il faut néanmoins réfléchir aux interfaces : il faut que chaque équipe sache sous quelle forme l'autre équipe va échanger des informations avec elle. Il faut définir des formats communs.
Avant de détailler ce point, il faut précisier que les données sont partagées par tout le monde et sont décrites dans des tables, il est préférable de préciser les colonnes importantes dans chacune d'entre elles. Par soucis de simplification, on ne s'intéresse qu'au second tour : la méthode ne s'adresse qu'à un des deux partis principaux et on fait l'hypothèse qu'ils sont majoritairement présents au second tour. Les informations proviennent de plusieurs sources mais elles recouvrent :

- *résultat des élections*
  - code département + code commune + code canton + bureau de vote = **identifiant bureau de vote**
  - **numéro circonscription**
  - inscrits
  - votants
  - exprimés
  - nuance du candidat
  - nombre de voix du candidat
- *les contours des circonscription*
  - **numéro circonscription**
  - contour (shape)
- *les contours des bureaux*
  - code département + code commune + code canton + bureau de vote = **identifiant bureau de vote**
  - contour (shape)

En gras, les champs qui serviront à faire des jointures ou à calculer des résultats. Nous pouvons maintenant définir les résultats de la méthode et une façon commune de décrire les informations dont on a besoin tout au long de la chaîne de traitement :

- **solution :** un dictionnaire `{ circonscription : [ liste des bureaux ] }`, c'est le résultat principal attendu. Nous pouvons facilement construire l'association actuelle. Nous voulons changer cette association pour favoriser un parti.

- **bureaux fontières et voisins :** un dictionnaire `{ bureau : [ liste des bureaux voisins ] }`, comme l'association bureaux / association va changer, il faut reconstruire les contours des circonscriptions

données manquantes
La location des bureaux de votes n'est pas disponible pour tous les bureaux de votes. On ne pourra déplacer que ceux qu'on sait localiser.

1.2 Données

On reprend les exemples de code fournis dans le notebook de l'énoncé.

- Résultat des élections législatives françaises de 2012 au niveau bureau de vote
- Countours des circonscriptions des législatives
- Localisation des buraux de votes
- Localisation des villes

[5]:
```
from actuariat_python.data import elections_legislatives_bureau_vote
tour = elections_legislatives_bureau_vote(source='xd')
tour["T2"].sort_values(["Code département", "N° de circonscription Lg"]).head(n=2)
```

[5]:
```
      N° tour Code département  Code de la commune Nom de la commune  \
3858       2               01                  16           Arbigny
3859       2               01                  16           Arbigny

      N° de circonscription Lg  N° de canton N° de bureau de vote  Inscrits  \
3858                         1            26                 0001       309
3859                         1            26                 0001       309

      Votants  Exprimés  N° de dépôt du candidat Nom du candidat  \
3858      146       144                       32          BRETON
3859      146       144                       33           DEBAT

     Prénom du candidat Code nuance du candidat  Nombre de voix du candidat
3858             Xavier                     UMP                         87
3859      Jean-François                     SOC                         57
```

[6]:
```
from actuariat_python.data import elections_legislatives_circonscription_geo
geo = elections_legislatives_circonscription_geo()
geo.sort_values(["department", "code_circonscription"]).head(n=2)
```

[6]:
```
   code_circonscription department  numero  \
11                01001         01       1
12                01002         01       2

                                              communes  \
11  01053-01072-01106-01150-01177-01184-01195-0124…
12  01008-01047-01099-01202-01213-01224-01366-0138…

                                             kml_shape  simple_form
11  <Polygon><outerBoundaryIs><LinearRing><coordin…        False
12  <Polygon><outerBoundaryIs><LinearRing><coordin…         True
```

[7]:
```
from actuariat_python.data import elections_vote_places_geo
bureau_geo = elections_vote_places_geo()
bureau_geo.head(n=2)
```

[7]:
```
        address   city  n            place   zip              full_address  \
0  cours verdun  bourg  1  salle des fêtes  1000  cours verdun 01000 bourg
1  cours verdun  bourg  2  salle des fêtes  1000  cours verdun 01000 bourg

    latitude  longitude                                        geo_address
0  46.206605   5.228364  Cours de Verdun, Le Peloux, Les Vennes, Bourg-…
1  46.206605   5.228364  Cours de Verdun, Le Peloux, Les Vennes, Bourg-…
```

1.3 Statistiques

[8]:
```
t2 = tour["T2"]
t2.columns
```

[8]:
```
Index(['N° tour', 'Code département', 'Code de la commune',
       'Nom de la commune', 'N° de circonscription Lg', 'N° de canton',
       'N° de bureau de vote', 'Inscrits', 'Votants', 'Exprimés',
       'N° de dépôt du candidat', 'Nom du candidat', 'Prénom du candidat',
       'Code nuance du candidat', 'Nombre de voix du candidat'],
      dtype='object')
```

Nous allons ajouter un identifiant pour les bureaux et les circonscriptions afin d'opérer facilement des fusions entre base de données plus facilement. Comme l'objectif est de changer les bureaux de vote de circonscription, le code de la circonscription ne peut pas être utilisé pour identifier un bureau de vote. Nous allons vérifier que cette hypothèse tient la route. Codes choisis :

- identifiant cironscription : **DDCCC#**, `D` pour code département, `C` pour code circonscription
- identifiant bureau de vote : **DDMMAABBB#**, `D` pour code département, `M` pour code commune, `A` pour code canton, `B` pour code bureau

[9]:
```
cols = ["Code département", "Code de la commune", "N° de canton", "N° de bureau de␣
 ↪vote"]
def code_bureau(dd, cc, aa, bb):
    bb = bb if isinstance(bb, str) else ("%03d" % bb)
    if len(bb) > 3: bb = bb[-3:]
    cc = cc if isinstance(cc, str) else ("%03d" % cc)
    aa = aa if isinstance(aa, str) else ("%02d" % aa)
    dd = dd if isinstance(dd, str) else ("%02d" % dd)
    # on ajoute un "#" à la fin pour éviter que pandas converisse la colonne numérique
    # et supprime les 0 devant l'identifiant
    return dd + cc + aa + bb + "#"
t2["idbureau"] = t2.apply(lambda row: code_bureau(*[row[c] for c in cols]), axis=1)
t2.head(n=2)
```

[9]:
```
   N° tour Code département  Code de la commune Nom de la commune  \
0        2               ZA                 101        Les Abymes
1        2               ZA                 101        Les Abymes

   N° de circonscription Lg  N° de canton N° de bureau de vote  Inscrits  \
0                         1             1                0001       477
```

```
1                          1             1                 0001       477

   Votants  Exprimés  N° de dépôt du candidat Nom du candidat  \
0      252       236                        9          JALTON
1      252       236                       17         DURIMEL

  Prénom du candidat Code nuance du candidat  Nombre de voix du candidat  \
0               Eric                     SOC                         182
1              Harry                     VEC                          54

      idbureau
0  ZA10101001#
1  ZA10101001#
```

[10]:
```
t2["idcirc"] = t2.apply(lambda row: str(row["Code département"]) + "%03d" % row["N° de␣
 ↪circonscription Lg"] + "#", axis=1)
t2.head(n=2)
```

[10]:
```
   N° tour Code département  Code de la commune Nom de la commune  \
0        2               ZA                 101        Les Abymes
1        2               ZA                 101        Les Abymes

   N° de circonscription Lg  N° de canton N° de bureau de vote  Inscrits  \
0                         1             1                 0001       477
1                         1             1                 0001       477

   Votants  Exprimés  N° de dépôt du candidat Nom du candidat  \
0      252       236                        9          JALTON
1      252       236                       17         DURIMEL

  Prénom du candidat Code nuance du candidat  Nombre de voix du candidat  \
0               Eric                     SOC                         182
1              Harry                     VEC                          54

      idbureau  idcirc
0  ZA10101001#  ZA001#
1  ZA10101001#  ZA001#
```

[11]:
```
len(set(t2["idcirc"]))
```

[11]:
```
541
```

541 < 577 est inférieur au nombre de députés. Cela signifie que 577 - 541 députés ont été élus au premier tour. Il faut aller récupérer les données du premier tour pour ces circonscriptions.

[12]:
```
t2circ = set(t2["idcirc"])
t1 = tour["T1"]
t1["idcirc"] = t1.apply(lambda row: str(row["Code département"]) + "%03d" % row["N° de␣
 ↪circonscription Lg"] + "#", axis=1)
t1["idbureau"] = t1.apply(lambda row: code_bureau(*[row[c] for c in cols]), axis=1)
t1["elu"] = t1["idcirc"].apply(lambda r: r not in t2circ)
t1nott2 = t1[t1["elu"]].copy()
t1nott2.head(n=2)
```

[12]:
```
     N° tour Code département  Code de la commune Nom de la commune  \
688        1               ZA                 104           Baillif
689        1               ZA                 104           Baillif

     N° de circonscription Lg  N° de canton N° de bureau de vote  Inscrits  \
688                         4            36                 0001       813
689                         4            36                 0001       813

     Votants  Exprimés  N° de dépôt du candidat Nom du candidat  \
688      386       357                       3          GUILLE
689      386       357                      18         MOLINIE

    Prénom du candidat Code nuance du candidat  Nombre de voix du candidat  \
688              Marc                      FN                           4
689             Louis                     DVD                           6

     idcirc     idbureau   elu
688  ZA004#  ZA10436001#  True
689  ZA004#  ZA10436001#  True
```

[13]:
```
len(set(t1nott2["idcirc"])) + 541
```

[13]:
```
577
```

Il ne reste plus qu'à les ajouter aux données du second tour.

[14]:
```
import pandas
t1t2 = pandas.concat([t1nott2, t2], axis=0, sort=True)
```

On compte le nombre d'Inscrits par bureaux de vote pour s'assurer que cela correspond à ce qui est attendu.

[15]:
```
statbu = t1t2[["Code département", "idcirc", "idbureau",
               "Inscrits"]].groupby(["Code département", "idcirc", "idbureau"],␣
 ↪as_index=False).max()
statbu.sort_values("Inscrits", ascending=False).head(n=5)
```

[15]:
```
      Code département  idcirc     idbureau  Inscrits
67921               ZZ  ZZ001#  ZZ00101001#    156645
67928               ZZ  ZZ008#  ZZ00808001#    109389
67926               ZZ  ZZ006#  ZZ00606001#    106689
67929               ZZ  ZZ009#  ZZ00909001#     97068
67924               ZZ  ZZ004#  ZZ00404001#     96964
```

[16]:
```
"nombre de bureaux de vote", statbu.shape[0]
```

[16]:
```
('nombre de bureaux de vote', 67932)
```

Le département ZZ ne correspond pas à un département connu et est étrangement plus grand que les autres. On vérifie.

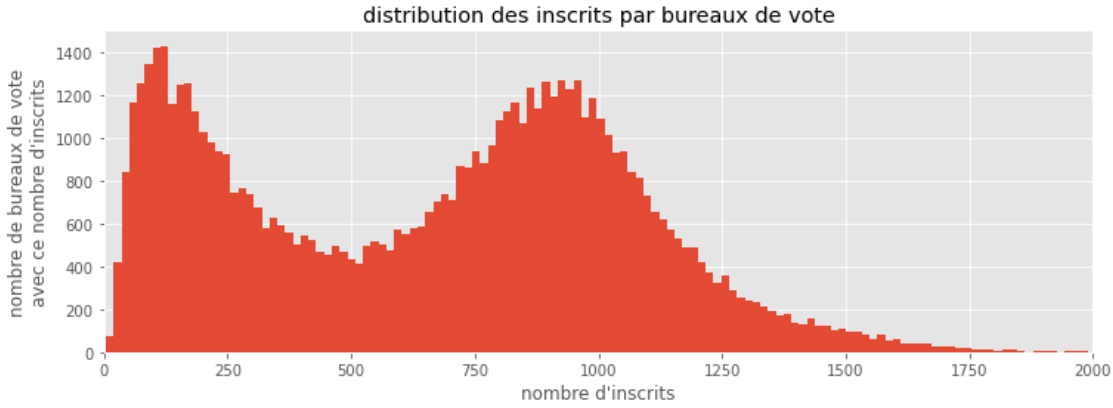
[17]:
```
t1t2[t1t2["Code département"] == "ZZ"].head(n=2)
```

[17]:
```
      Code de la commune Code département Code nuance du candidat  Exprimés  \
3789                   1              ZZ                     UMP     29223
3790                   1              ZZ                     SOC     29223
```

```
      Inscrits Nom de la commune Nom du candidat  Nombre de voix du candidat  \
3789    156645  Amérique du Nord        LEFEBVRE                       13441
3790    156645  Amérique du Nord     NARASSIGUIN                       15782

     N° de bureau de vote  N° de canton  N° de circonscription Lg  \
3789                 0001             1                         1
3790                 0001             1                         1

      N° de dépôt du candidat  N° tour Prénom du candidat  Votants  elu  \
3789                      117        2           Frédéric    29869  NaN
3790                      143        2            Corinne    29869  NaN

         idbureau  idcirc
3789  ZZ00101001#  ZZ001#
3790  ZZ00101001#  ZZ001#
```

Ce bureau de vote n'est pas en France. Il est bien plus volumineux que les autres et nous ne pourrons pas le rapprocher géographiquement des autres. On n'en tiendra plus compte. On enlève également les bureaux de vote qui commencent par `Z` (`ZA`, `ZZ`, `ZW`).

[18]:
```
fig, ax = plt.subplots(figsize=(12,4))
statbu[(statbu["Code département"] != "ZZ") & (statbu["Code département"] != "ZW")].
↪hist( \
       bins=200, ax=ax)
ax.set_xlim(0, 2000)
ax.set_xlabel("nombre d'inscrits")
ax.set_ylabel("nombre de bureaux de vote\navec ce nombre d'inscrits")
ax.set_title("distribution des inscrits par bureaux de vote");
```

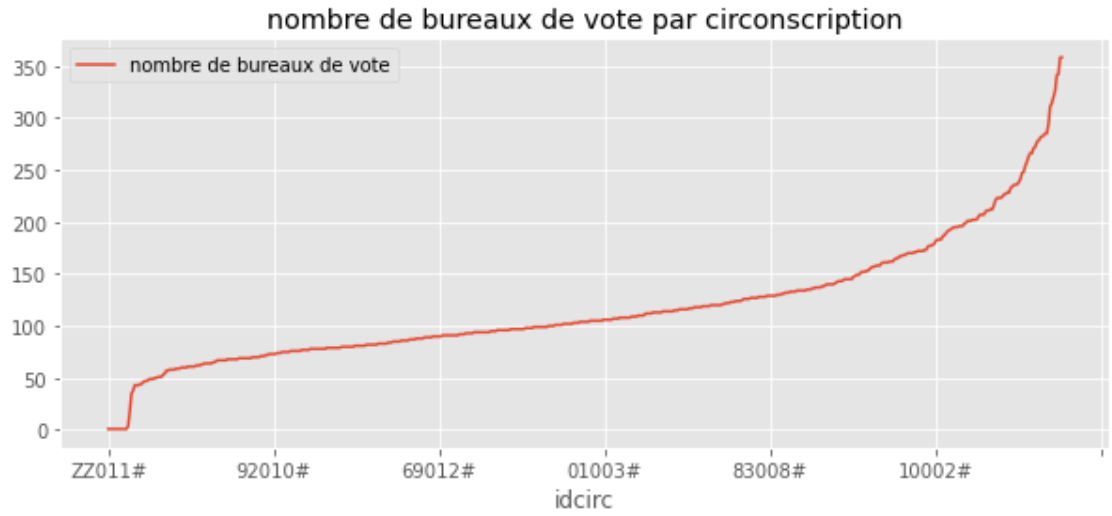


[19]:
```
nbbur = statbu[["idcirc","idbureau"]].groupby("idcirc").count()
nbbur.columns=["nombre de bureaux de vote"]
nbbur.head()
```

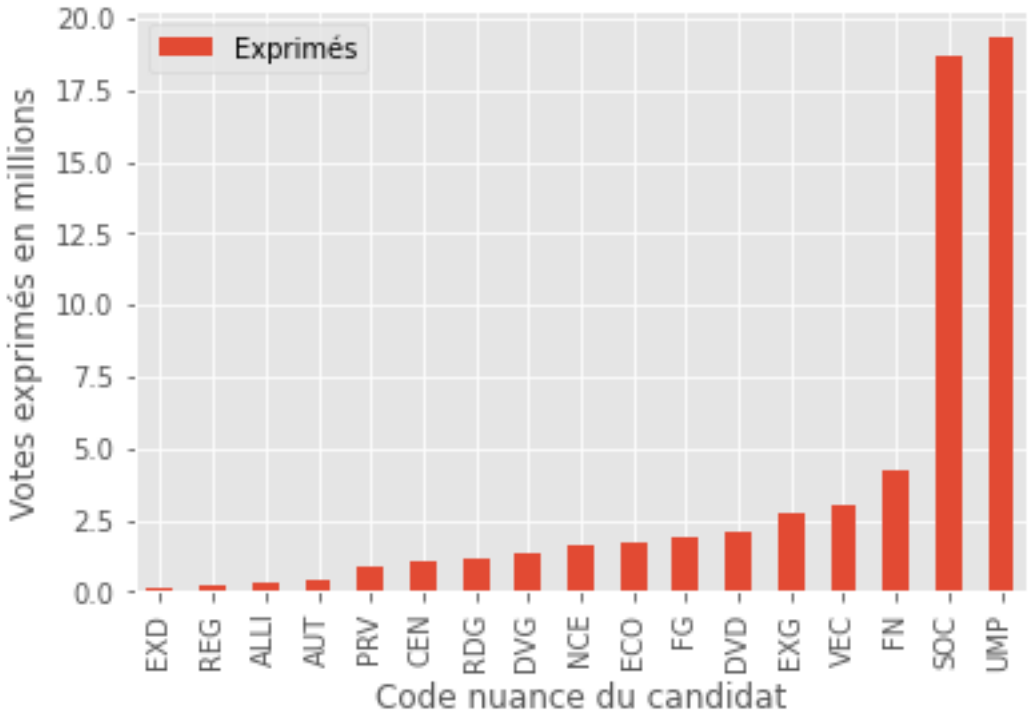
[19]:
```
         nombre de bureaux de vote
idcirc
01001#                         117
01002#                          98
01003#                         106
```

```
01004#                          113
01005#                          145
```

[20]:
```
ax = nbbur[nbbur[nbbur.columns[0]] > 0].sort_values(nbbur.columns[0]).
↪plot(figsize=(10,4))
ax.set_title("nombre de bureaux de vote par circonscription");
```



[21]:
```
t1t2noz = t1t2[t1t2["Code département"].apply(lambda r: not r.startswith("Z"))]
elu = t1t2noz[["Code nuance du candidat", "Exprimés"]].groupby("Code nuance du␣
↪candidat").sum()
ax = (elu.sort_values("Exprimés")/1000000).plot(kind="bar")
ax.set_ylabel("Votes exprimés en millions");
```



Ces statistiques correspondent à ce qui est attendu. Avec une centaine de bureau par circonscription, nous devrions pouvoir changer la répartition.

1.4 Identifiants

Nous aurons besoin de croiser les données provenant de plusieurs bases en fonction des circonscriptions et des bureaux de votes. Il convient de déterminer ce qui identifient de façon unique un bureau de vote et une circonscription. Il faut se rappeler des conventions choisies :

- identifiant cironscription : **DDCCC#**, `D` pour code département, `C` pour code circonscription
- identifiant bureau de vote : **DDMMAABBB#**, `D` pour code département, `M` pour code commune, `A` pour code canton, `B` pour code bureau

Le caractère `#` sert à éviter la conversion automatique d'une colonne au format numérique par pandas.

[22]:
```
t1t2.head(n=2)
```

[22]:
```
     Code de la commune Code département Code nuance du candidat  Exprimés  \
688                 104              ZA                      FN       357
689                 104              ZA                     DVD       357

     Inscrits Nom de la commune Nom du candidat  Nombre de voix du candidat  \
688       813           Baillif          GUILLE                          4
689       813           Baillif         MOLINIE                          6

    N° de bureau de vote  N° de canton  N° de circonscription Lg  \
688                 0001            36                        4
689                 0001            36                        4
```

```
     N° de dépôt du candidat  N° tour Prénom du candidat  Votants   elu  \
688                        3        1               Marc      386  True
689                       18        1              Louis      386  True

        idbureau  idcirc
688  ZA10436001#  ZA004#
689  ZA10436001#  ZA004#
```

[23]:
```
len(set(t1t2["idcirc"]))
```

[23]:
```
577
```

Le nombre de circsonscriptions est le nombre attendu. On vérifie que les circonscriptions ne s'étendant pas sur plusieurs départements. Cela signifie que nous pouvons optimiser les répartitions des bureaux par département de façon indépendantes.

[24]:
```
t1t2.groupby(["idcirc", "Code département"], as_index=False).count().shape
```

[24]:
```
(577, 18)
```

La ligne suivante montre qu'une circonscription englobe plusieurs cantons.

[25]:
```
t1t2.groupby(["idcirc", "N° de canton"], as_index=False).count().shape
```

[25]:
```
(4171, 18)
```

Combien y a-t-il de bureaux de vote ?

[26]:
```
len(set(t1t2["idbureau"]))
```

[26]:
```
67932
```

On vérifie que nous n'avons pas plusieurs le même nom de bureaux pour une même circonscription auquel cas cela voudrait dire que l'identifiant choisi n'est pas le bon.

[27]:
```
t1t2.groupby(["idcirc", "idbureau"], as_index=False).count().shape
```

[27]:
```
(67932, 18)
```

Même nombre. Tout va bien !

1.5 Evaluation d'une solution

Dans la suite, on se sert des deux colonnes `idbureau` et `idcirc` comme identifiant de bureaux et circonscription et on s'intéresse à une association *circonscription - bureau* quelconque.

[28]:
```
import numpy

def agg_circonscription(data_vote, solution=None, col_circ="idcirc",
                        col_place="idbureau", col_vote="Nombre de voix du candidat",
                        col_nuance="Code nuance du candidat"):
    """
    Calcul la nuance gagnante dans chaque circonscription.

    @param      data_vote    dataframe pour les voix
    @param      solution     dictionnaire ``{ circonscription : liste de bureaux }``,
                             si None, la fonction considère la solution officielle
    @param      col_circ     colonne contenant la circonscription (si solution = None)
```

```
    @param      col_place     colonne contenant l'identifiant du bureaux de votes
    @param      col_vote      colonne contenant les votes
    @param      col_nuance    colonne contenant le parti ou la nuance
    @return                   matrice de résultats, une ligne par circoncription, une␣
↪colonne par nuance/parti
    """
    if solution is None:
        # on reprend l'association circoncscription - bureau de la dernière élection
        agg = data_vote[[col_circ, col_nuance, col_vote]].groupby([col_circ,␣
↪col_nuance], as_index=False).sum()
    else:
        # on construit la nouvelle association
        rev = {}
        for k, v in solution.items():
            for place in v:
                if place in rev:
                    raise ValueError("Un bureaux est associé à deux circonscriptions :␣
↪{0}".format([rev[place], k]))
                rev[place] = k
        keep = data_vote[[col_place, col_vote, col_nuance]].copy()
        if col_circ is None:
            col_circ = "new_circ_temp"
        keep[col_circ] = keep[col_place].apply(lambda r: rev[r])
        agg = keep[[col_circ, col_nuance, col_vote]].groupby([col_circ, col_nuance],␣
↪as_index=False).sum()

    # les données sont maintenant agrégées par circonscription, il faut déterminer le␣
↪gagnant
    piv = agg.pivot(col_circ, col_nuance, col_vote)
    gagnant = []
    votes = []
    sums = []

    for row in piv.values:
        mx = max((r, i) for i, r in enumerate(row) if not numpy.isnan(r))
        gagnant.append(piv.columns[mx[1]])
        votes.append(mx[0])
        sums.append(sum(r for r in row if not numpy.isnan(r)))

    piv["winner"] = gagnant
    piv["nbwinner"] = votes
    piv["total"] = sums
    return piv

score = agg_circonscription(t1t2noz)
score.head()
```

[28]:

Code nuance du candidat	ALLI	AUT	CEN	DVD	DVG	ECO	EXD	EXG	FG
idcirc									
01001#	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
01002#	NaN	NaN	NaN	NaN	19529.0	NaN	NaN	NaN	NaN
01003#	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
01004#	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
01005#                    NaN  NaN  NaN  NaN      NaN  NaN  NaN  NaN NaN

Code nuance du candidat      FN  NCE  PRV      RDG  REG      SOC      UMP  \
idcirc
01001#                      NaN  NaN  NaN      NaN  NaN  22743.0  24233.0
01002#                   8530.0  NaN  NaN      NaN  NaN      NaN  22327.0
01003#                      NaN  NaN  NaN      NaN  NaN  15653.0  19266.0
01004#                      NaN  NaN  NaN  19780.0  NaN      NaN  26175.0
01005#                      NaN  NaN  NaN      NaN  NaN  17012.0  22008.0

Code nuance du candidat  VEC winner  nbwinner    total
idcirc
01001#                   NaN    UMP   24233.0  46976.0
01002#                   NaN    UMP   22327.0  50386.0
01003#                   NaN    UMP   19266.0  34919.0
01004#                   NaN    UMP   26175.0  45955.0
01005#                   NaN    UMP   22008.0  39020.0
```

[29]:
```
score.shape
```

[29]:
```
(539, 20)
```

[30]:
```
len(set(t1t2noz["idcirc"]))
```

[30]:
```
539
```

[31]:
```
len(set(t1t2["idcirc"]))
```

[31]:
```
577
```

Le processus ne s'appliquera qu'aux circonscriptions de la métropole, soit 565. Le résultat d'une nouvelle répartition peut être calculée comme ceci :

[32]:
```
count = score[["winner", "nbwinner"]].groupby(["winner"]).count()
count.sort_values("nbwinner", ascending=False)
```

[32]:
```
Code nuance du candidat  nbwinner
winner
SOC                          263
UMP                          190
DVG                           16
VEC                           16
NCE                           12
RDG                           11
FG                            10
DVD                            9
PRV                            6
ALLI                           2
FN                             2
CEN                            1
EXD                            1
```

Le parti socialiste à 263 députés. L'UMP 190. Nous allons essayé de changer ces nombres.

1.6 Visualisation d'une solution

1.6.1 Première carte : circonscriptions actuelles

On reprend la même signature que la fonction précédente avec le dataframe `geo` qui contient la définition des circonscriptions. On commence par créer une fonction qui extrait les contours qui sont disponibles sous formes de chaînes de caractères. Le résultat est inspiré de ce notebook. Tous les résultats que nous allons construire vont être proches de ce résultat. Cela permettra de vérifier que nous nous trompons pas au moment où nous allons visualiser les nouvelles circonscriptions.

[33]:
```
def process_boundary(bound_string):
    ext = bound_string.split("<coordinates>")[-1].split("</coordinates>")[0]
    spl = ext.split(" ")
    return [(float(ll[0]), float(ll[1])) for ll in [_.split(",") for _ in spl]]

s = """
    <Polygon><outerBoundaryIs><LinearRing><coordinates>5.294455999999968,46.193934 5.
 ↪279780999999957,46.201967</coordinates></LinearRing></outerBoundaryIs></Polygon>
    """
r = process_boundary(s)
r
```

[33]:
```
[(5.294455999999968, 46.193934), (5.279780999999957, 46.201967)]
```

Certaines circonscriptions n'ont pas de contours.

[34]:
```
geo[geo.code_circonscription=="98702"]
```

[34]:

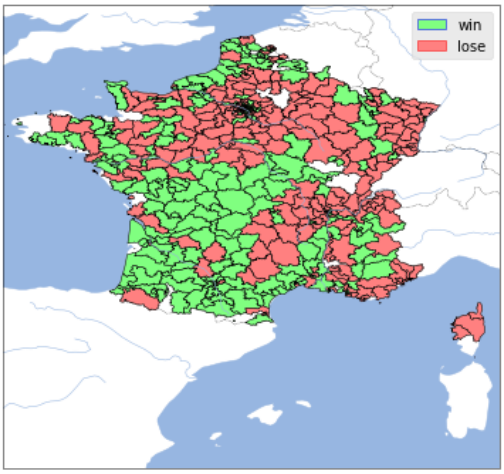
	code_circonscription	department	numero	communes	kml_shape	simple_form
575	98702	987	2	NaN	NaN	True

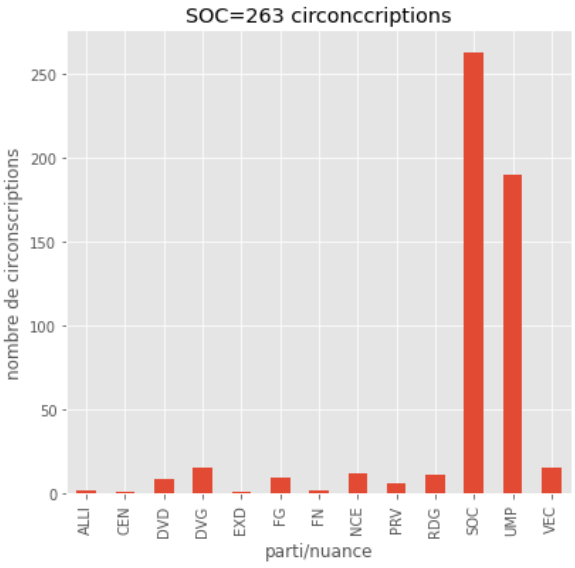
La fonction suivante projette les circonscription existantes car on ne sait pas encore construire le contour d'une circonscription construite à partir d'une solution.

[35]:
```
import numpy
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from shapely.geometry import Polygon
import geopandas
from matplotlib.patches import Patch

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


def agg_circonscription_viz(thewinner, geo, data_vote,
                col_circ="idcirc", col_place="idbureau", col_vote="Nombre de voix du␣
 ↪candidat",
                col_nuance="Code nuance du candidat", figsize=(14,6), **kwargs):
```

```
    """
    Visualise la nuance gagnante dans chaque circonscription.

    @param      thewinner    parti qu'on souhaite influencer
    @param      geo          shapes pour chaque circonscription
    @param      axes         None ou deux systèmes d'axes
    @param      figsize      dimension du graphiques
    @param      kwargs       options additionnelles

    @param      data_vote    dataframe de type
    @param      col_circ     colonne contenant la circonscription (si solution = None)
    @param      col_place    colonne contenant l'identifiant du bureaux de votes
    @param      col_vote     colonne contenant les votes
    @param      col_nuance   colonne contenant le parti ou la nuance
    @return                  matrice de resultat, une ligne par circoncription, une␣
↪colonne par nuance/parti
    """
    # on transforme les dataframes en dictionnaires
    score = agg_circonscription(data_vote, col_circ=col_circ, col_place=col_place,
                                col_vote=col_vote, col_nuance=col_nuance)
    winner = score[["winner"]].to_dict("index")
    shapes = geo.set_index("code_circonscription")[["kml_shape"]].to_dict("index")

    fig = plt.figure(figsize=figsize)

    ax1 = fig.add_subplot(1, 2, 1, projection=ccrs.PlateCarree())
    ax1.set_extent([-5, 10, 38, 52])
    ax1.add_feature(cfeature.OCEAN.with_scale('50m'))
    ax1.add_feature(cfeature.RIVERS.with_scale('50m'))
    ax1.add_feature(cfeature.BORDERS.with_scale('50m'), linestyle=':')

    ax2 = fig.add_subplot(1, 2, 2)
    axes = [ax1, ax2]

    # on dessine la distribution des circonscriptions
    count = score[["winner", "nbwinner"]].groupby(["winner"]).count()
    count.sort_values("nbwinner", ascending=False)
    count.plot(ax=axes[1], kind="bar", legend=False)
    axes[1].set_xlabel("parti/nuance")
    axes[1].set_ylabel("nombre de circonscriptions")

    # on calcule le nombre de places le parti considéré
    count = count.reset_index(drop=False)
    count["iswin"] = count["winner"] == thewinner
    ratio = count[["nbwinner", "iswin"]].groupby("iswin").sum().sort_index()
    nbcirc = ratio.iloc[1,0]
    axes[1].set_title("{0}={1} circonccriptions".format(thewinner, nbcirc))

    polys = []
    colors = []

    for circ, vals in shapes.items():
        if circ.startswith("Z"):
```

```
            # outside
            continue
        if not circ.endswith("#"):
            # nous avons ajouté un dièse
            circ += "#"
        shape = vals["kml_shape"]
        if isinstance(shape, float):
            # NaN
            continue

        geo_points = process_boundary(shape)
        # geo_points = [lambert932WGPS(x,y) for x, y in geo_points]

        if circ in winner:
            win = winner[circ]["winner"]
            color = (0.5, 1.0, 0.5) if win == thewinner else (1.0, 0.5, 0.5)
        else:
            color = "black"

        if len(geo_points) < 4:
            continue
        poly = Polygon(geo_points)
        polys.append(poly)
        colors.append(color)

    data = geopandas.GeoDataFrame(dict(geometry=polys, colors=colors))
    geopandas.plotting.plot_polygon_collection(axes[0], data['geometry'],␣
 ↪facecolor=data['colors'],
                                               values=None, edgecolor='black')

    legend_elements = [Patch(facecolor=(0.5, 1.0, 0.5), edgecolor='b', label='win'),
                       Patch(facecolor=(1.0, 0.5, 0.5), edgecolor='r', label='lose')]
    axes[0].legend(handles=legend_elements, loc='upper right')
    return fig, axes

fig, axes = agg_circonscription_viz("SOC", geo, t1t2noz)
fig;
```





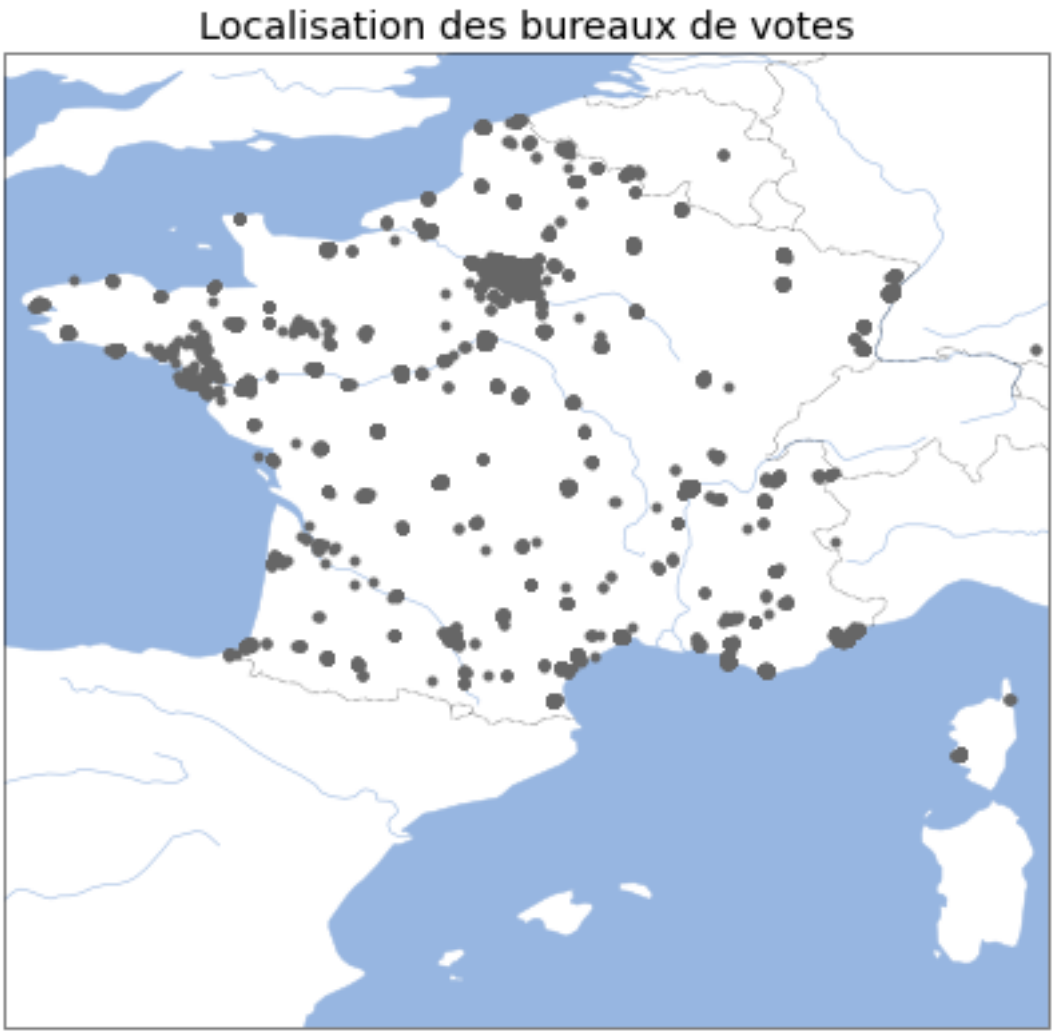
Certaines parties du territoires manquent. Les contours manquent ou les résultats manquent pour une certaine circonscription. La cohérence des données devraient être vérifiées car celles-ci viennent de sources différentes.

1.6.2 Dessiner de nouvelles circonscriptions ?

Si on change les circonscriptions, les contours des anciennes circonscriptions ne sont plus valables ! Si on ne dispose que de la position des bureaux de vote, il faut reconstruire le contour de chaque circonscription en fonction de la position des bureaux de vote. La méthode : construire un graphe de Voronoï et ne garder que les frontières entre bureaux de circonscriptions différentes. Si on dispose de contours pour chaque bureau de vote, l'autre option consiste à fusionner ces contours en éliminant la surface commune. C'est ce que fait la fonction cascaded_union du module shapely.
Le problème principal devient l'association de la location des bureaux de vote avec les résultats des votes. Tout d'abord nous avons besoin de vérifier que nous avons suffisamment de bureaux de vote localisé dans la base *bureau_geo* et on s'aperçoit que c'est largement insuffisant.

[36]:
```
cols = ["city", "zip", "n"]
bureau_geo["idbureaugeo"] = bureau_geo.apply(lambda row: "-".join(str(row[_]) for _ in␣
 ↪cols), axis=1)
```

[37]:
```
ax = carte_france()
lons = bureau_geo["longitude"]
lats = bureau_geo["latitude"]
ax.plot(lons, lats, ".", color=(0.4, 0.4, 0.4))
ax.set_title("Localisation des bureaux de votes");
```



1.6.3 Autres sources pour les bureaux de votes

La faible densité des bureaux de votes oblige à changer de jeu de données et d'utiliser celui de cartelec utilisé pour l'année 2007. Il devrait en grande majorité valable pour l'année 2012 mais nous allons observer que la base n'est pas telle qu'on pourrait s'attendre.

[38]:
```
from pyensae.datasource import download_data
shp_vote = download_data("base_cartelec_2007_2010.zip")
import shapefile
rshp = shapefile.Reader("fond0710.shp", encoding="utf-8", encodingErrors="ignore")
shapes = rshp.shapes()
records = rshp.records()
```

[39]:
```
{k[0]:v for k,v in zip(rshp.fields[1:], records[0])}, shapes[0].points[:5]
```

[39]:
```
({'BUREAU': '01001',
  'CODE': '01001',
  'NOM': "L'Abergement-Clmenciat",
```

```
 'CODEARRT': '012',
 'CODEDEP': '01',
 'CODEREG': '82',
 'CODECANT': '10',
 'CANTON': 'CHATILLON-SUR-CHALARONNE',
 'CIRCO': '04'},
[(846774.7025280485, 6563840.655779875),
 (847430.4726776106, 6566444.631470905),
 (848975.0615885032, 6566530.102978201),
 (849532.5253064571, 6565971.4588501565),
 (848969.0813380895, 6564398.911644492)])
```

[40]:
```
shapes[0].__dict__
```

[40]:
```
{'shapeType': 5,
 'points': [(846774.7025280485, 6563840.655779875),
  (847430.4726776106, 6566444.631470905),
  (848975.0615885032, 6566530.102978201),
  (849532.5253064571, 6565971.4588501565),
  (848969.0813380895, 6564398.911644492),
  (850941.7401535356, 6563209.5425065085),
  (849896.4212796891, 6562719.844144765),
  (849632.2745031306, 6561522.415193593),
  (849891.0276243397, 6560738.406460746),
  (848732.0257644501, 6559575.068823495),
  (848585.9032087281, 6560169.582690463),
  (847664.0345600601, 6560616.395794825),
  (847793.2580021, 6562243.125831007),
  (846774.7025280485, 6563840.655779875)],
 'parts': [0],
 'bbox': [846774.7025280485, 6559575.068823495, 850941.7401535356,
6566530.102978201]}
```

Les coordonnées ne sont pas des longitudes et latitudes. Il faut les convertir.

1.6.4 Conversion des coordoonnées et identifiant de bureau

Voir ce notebook. La fonction qui suit est assez longue et elle est exécutée un grand nombre de fois. Dans notre cas, le traitement n'est pas encore trop long et n'est exécuté qu'une fois autrement il faudrait l'accélérer avec numba ou cython.

[41]:
```
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
```

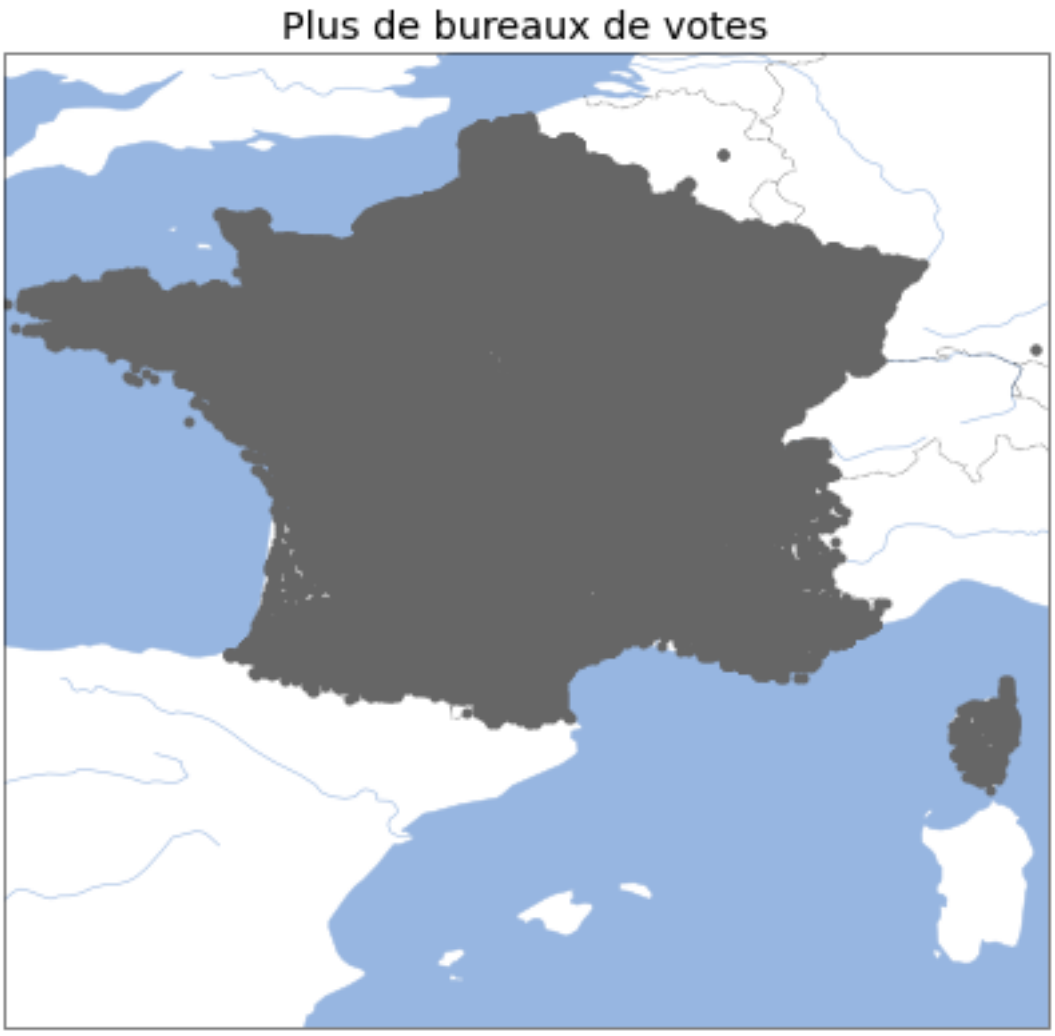
```
    sinPhiit0 = math.tanh(latiso + constantes.GRS80E * math.atanh(constantes.GRS80E *␣
 ↪math.sin(1)))
    sinPhiit1 = math.tanh(latiso + constantes.GRS80E * math.atanh(constantes.GRS80E *␣
 ↪sinPhiit0))
    sinPhiit2 = math.tanh(latiso + constantes.GRS80E * math.atanh(constantes.GRS80E *␣
 ↪sinPhiit1))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

[41]:
```
((-4.1615802638173065, 41.303505287589545),
 (10.699505053975292, 50.85243395553585))
```

[42]:
```
for shape in shapes:
    x1, y1 = lambert932WGPS(shape.bbox[0], shape.bbox[1])
    x2, y2 = lambert932WGPS(shape.bbox[2], shape.bbox[3])
    shape.bbox = [x1, y1, x2, y2]
    shape.points = [lambert932WGPS(x,y) for x,y in shape.points]
```

On vérifie que nous disposons de beaucoup plus de bureaux de vote localisés.

[43]:
```
ax = carte_france()
lons = bureau_geo["longitude"]
lats = bureau_geo["latitude"]
ax.plot(lons, lats, ".", color=(0.4, 0.4, 0.4))
ax.set_title("Plus de bureaux de votes");

lons = []
lats = []
for shape in shapes:
    x1, y1, x2, y2 = shape.bbox
    x = (x1+x2) / 2
    y = (y1+y2) / 2
    lons.append(x)
    lats.append(y)
ax.plot(lons, lats, ".", color=(0.4, 0.4, 0.4));
```



La France est recouverte de gris. La densité des bureaux de votes est plus conforme à celle attendue. La conversion des coordonnées a fonctionné et les données seront exploitables.

[44]:
```
len(shapes), len(set(t1t2noz.idbureau))
```

[44]:
```
(50578, 65717)
```

C'est quand même moins que les 67920 bureaux de vote enregistrés dans la table des élections ! Il y a 17000 bureaux de votes que nous ne pouvons pas localiser. On essaye un bureau au hasard pour deviner le sens des informations fournies dans les *records* :

[45]:
```
{k[0]:v for k,v in zip(rshp.fields[1:], records[11000])}
```

[45]:
```
{'BUREAU': '25038',
 'CODE': '25038',
 'NOM': 'Avilley',
 'CODEARRT': '251',
 'CODEDEP': '25',
 'CODEREG': '43',
 'CODECANT': '23',
```

```
 'CANTON': 'ROUGEMONT',
 'CIRCO': '03'}
```

[46]:
```
t1t2[t1t2["Nom de la commune"] == "Avilley"]
```

[46]:
```
       Code de la commune Code département Code nuance du candidat  Exprimés  \
29979                 38              25                     UMP        96
29980                 38              25                     SOC        96

       Inscrits Nom de la commune Nom du candidat  Nombre de voix du candidat  \
29979       153           Avilley          BONNOT                         60
29980       153           Avilley         MARTHEY                         36

      N° de bureau de vote  N° de canton  N° de circonscription Lg  \
29979                 0001            23                         3
29980                 0001            23                         3

       N° de dépôt du candidat  N° tour Prénom du candidat  Votants  elu  \
29979                       47        2             Marcel       98  NaN
29980                       50        2             Arnaud       98  NaN

          idbureau  idcirc
29979  2503823001#  25003#
29980  2503823001#  25003#
```

Où est le code du bureau ?

[47]:
```
t1t2[t1t2["Nom de la commune"] == "Nouzonville"]
```

[47]:
```
       Code de la commune Code département Code nuance du candidat  Exprimés  \
12229                 328              08                     SOC       755
12230                 328              08                     UMP       755
12231                 328              08                     SOC       711
12232                 328              08                     UMP       711
12233                 328              08                     SOC       571
12234                 328              08                     UMP       571

       Inscrits Nom de la commune Nom du candidat  Nombre de voix du candidat  \
12229      1707       Nouzonville         LEONARD                        486
12230      1707       Nouzonville        RAVIGNON                        269
12231      1565       Nouzonville         LEONARD                        481
12232      1565       Nouzonville        RAVIGNON                        230
12233      1149       Nouzonville         LEONARD                        357
12234      1149       Nouzonville        RAVIGNON                        214

      N° de bureau de vote  N° de canton  N° de circonscription Lg  \
12229                 0001            34                         2
12230                 0001            34                         2
12231                 0002            34                         2
12232                 0002            34                         2
12233                 0003            34                         2
12234                 0003            34                         2

       N° de dépôt du candidat  N° tour Prénom du candidat  Votants  elu  \
12229                       18        2         Christophe      782  NaN
```

```
12230                    27        2             Boris      782  NaN
12231                    18        2         Christophe     732  NaN
12232                    27        2             Boris      732  NaN
12233                    18        2         Christophe     590  NaN
12234                    27        2             Boris      590  NaN

          idbureau  idcirc
12229  0832834001#  08002#
12230  0832834001#  08002#
12231  0832834002#  08002#
12232  0832834002#  08002#
12233  0832834003#  08002#
12234  0832834003#  08002#
```

[48]:
```
[ {k[0]:v for k,v in zip(rshp.fields[1:], rec)} for rec in records if "Nouzonville" in␣
↪rec]
```

[48]:
```
[{'BUREAU': '08328',
  'CODE': '08328',
  'NOM': 'Nouzonville',
  'CODEARRT': '081',
  'CODEDEP': '08',
  'CODEREG': '21',
  'CODECANT': '34',
  'CANTON': 'NOUZONVILLE',
  'CIRCO': '02'}]
```

On regarde à Paris.

[49]:
```
[ {k[0]:v for k,v in zip(rshp.fields[1:], rec)} for rec in records if "Paris-10" in␣
↪rec][0:2]
```

[49]:
```
[{'BUREAU': '75110_1001',
  'CODE': '75110',
  'NOM': 'Paris 10e  arrondiss',
  'CODEARRT': '751',
  'CODEDEP': '75',
  'CODEREG': '11',
  'CODECANT': '24',
  'CANTON': 'Paris-10',
  'CIRCO': '05'},
 {'BUREAU': '75110_1002',
  'CODE': '75110',
  'NOM': 'Paris 10e  arrondiss',
  'CODEARRT': '751',
  'CODEDEP': '75',
  'CODEREG': '11',
  'CODECANT': '24',
  'CANTON': 'Paris-10',
  'CIRCO': '05'}]
```

Cela signifie que les bureaux de vote sont regroupées sur les petites villes et pas sur les grandes. Combien avons nous de bureaux de vote uniques et localisés ?

[50]:
```
len([ _ for _ in [ {k[0]:v for k,v in zip(rshp.fields[1:], rec)} for rec in records]␣
 ↪if "_" in _["BUREAU"]])
```

[50]: 15837

On peut maintenant reconstruire un identifiant de bureau, complet quand le bureau de vote est présent, incomplet quand il ne l'est pas.

[51]:
```
def shape_idbureau(rec):
    # département + commune + canton + bureau
    if "_" in rec["BUREAU"]:
        bb = "0" + rec["BUREAU"].split("_")[-1][-2:]
    else:
        bb = "***"
    return rec["CODEDEP"] + rec["CODE"][-3:] + rec["CODECANT"] + bb + "#"

shape_idbureau({k[0]:v for k,v in zip(rshp.fields[1:], records[11000])})
```

[51]: '2503823***#'

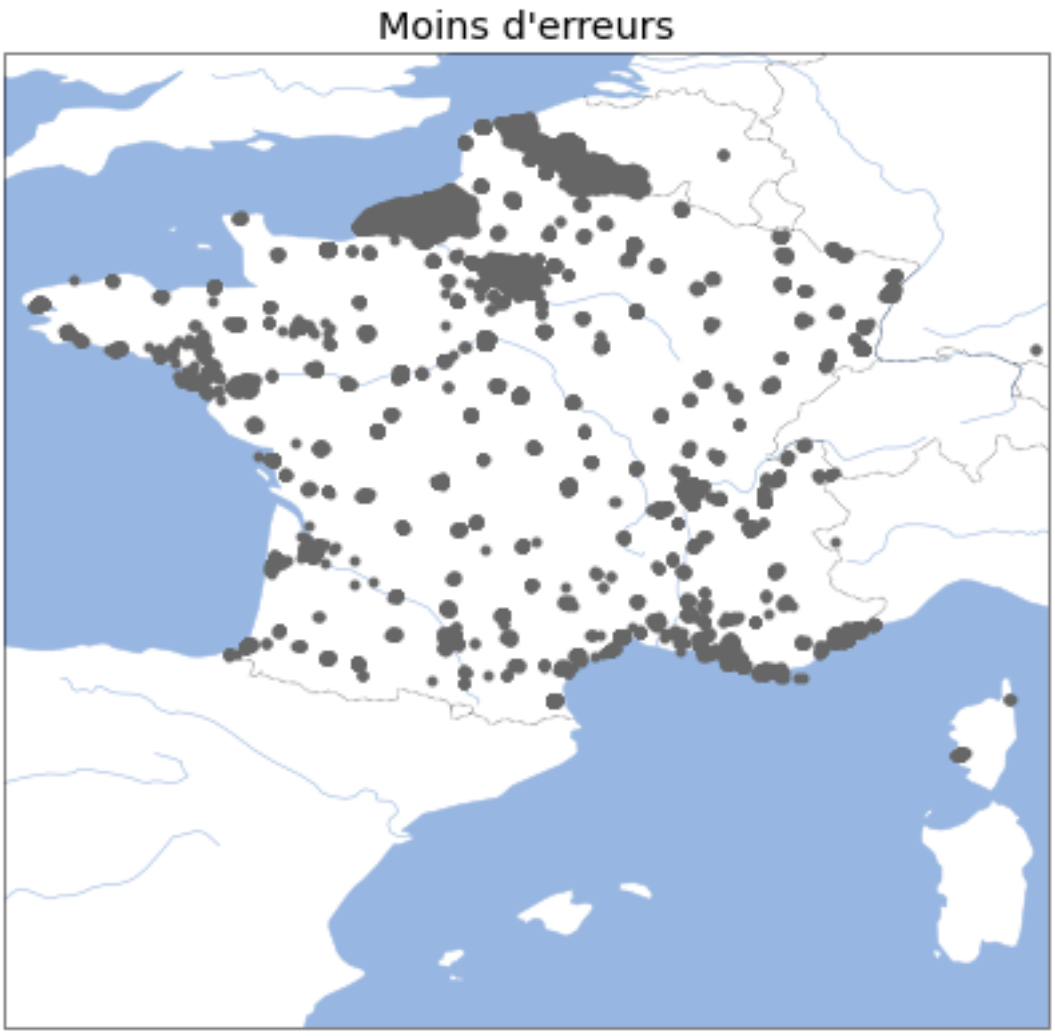
1.6.5 Implications sur la méthode globale

En résumé, nous avons :

- 67920 bureaux de vote en métropole
- 50578 lieux distincts
- 15837 bureaux de vote clairement identifiés et localisés
- ~35000 lieux qui correspondent au regroupement de bureaux de vote

Pour poursuivre, il va falloir agréger les résultats pour les bureaux de vote qui ont été regroupés dans la base qui fournit leur coordonnées ou tout simplement donner à ces bureaux de vote un identifiant unique. La seconde option, même si elle impose de conserver plus de données à l'avantage d'être plus simple et donc de générer moins d'erreur.

[52]:
```
ax = carte_france()
lons = bureau_geo["longitude"]
lats = bureau_geo["latitude"]
ax.plot(lons, lats, ".", color=(0.4, 0.4, 0.4))
ax.set_title("Moins d'erreurs");

lons = []
lats = []
for rec, shape in zip(records, shapes):
    d = {k[0]:v for k,v in zip(rshp.fields[1:], rec)}
    if "_" not in d["BUREAU"]:
        # bureau de vote pas unique
        continue
    x1, y1, x2, y2 = shape.bbox
    x = (x1+x2) / 2
    y = (y1+y2) / 2
    lons.append(x)
    lats.append(y)
ax.plot(lons, lats, ".", color=(0.4, 0.4, 0.4));
```



Clairement les grandes et moyennes villes.

[53]:
```
t1t2noz[list(_ for _ in t1t2noz.columns if "Code" in _ or "id" in _ or "N°" in _)].
↪head(n=2)
```

[53]:
```
       Code de la commune Code département Code nuance du candidat  \
47442                   4              06                      EXG
47443                   4              06                       FN

      Nom du candidat  Nombre de voix du candidat N° de bureau de vote  \
47442          PETARD                           0                 0101
47443            VIOT                          33                 0101

       N° de canton  N° de circonscription Lg  N° de dépôt du candidat  \
47442             1                         7                        8
47443             1                         7                       24

       N° tour Prénom du candidat     idbureau  idcirc
```

```
47442        1          Christian  0600401101#  06007#
47443        1           Mathilde  0600401101#  06007#
```

On choisit maintenant de remplacer les valeurs de la colonne `idbureau`, si le code `0600401101#` n'a pas de localisation connue, cela signifie qu'il est probablement agrégé avec d'autres bureaux de vote. On le remplace par `0600401***#`. Nous verrons cela plus bas.

1.6.6 Fusionner les shapefiles

C'est maintenant qu'on va utiliser la fonction cascade_union du module shapely. On extrait un sous-ensemble de bureaux de vote pour tester la fontion.

[54]:
```
canton04 = []
for rec, shape in zip(records, shapes):
    d = {k[0]:v for k,v in zip(rshp.fields[1:], rec)}
    if d["CODECANT"] == '10' and d['CODEDEP'] == '01':
        canton04.append((rec, shape))
len(canton04)
```

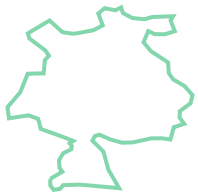
[54]:
```
16
```

[55]:
```
from random import randint
colors = ['#%06X' % randint(0, 0xAAAAAA) for i in range(len(canton04))]

def format_popud(d):
    key = ["CANTON", "BUREAU"]
    rows = ["{0}: {1}".format(k, d[k]) for k in key]
    pattern = "{0}".format("<br/>".join(rows))
    return pattern

import folium
c = canton04[0][1]
map_osm = folium.Map(location=[c.bbox[1], c.bbox[0]])
i = 0
for rec, shape in canton04:
    d = {k[0]:v for k,v in zip(rshp.fields[1:], rec)}
    map_osm.add_child(folium.PolyLine(locations=[(_[1], _[0]) for _ in shape.points],
                                      color=colors[i], popup=format_popud(d)))
    i += 1
from pyensae.notebookhelper import folium_html_map
folium_html_map(map_osm, width="50%")
```

[55]:
```
<pyensae.notebookhelper.folium_helper.folium_html_map.<locals>.CustomFoliumMap
at 0x1cb4e265518>
```

[56]:
```
from shapely.geometry import Point, Polygon
from shapely.ops import cascaded_union
polys = []
for rec, shape in canton04:
    poly = Polygon([(x,y) for x,y in shape.points])
    polys.append(poly)
union = cascaded_union(polys)
union.boundary
```

[56]:



[57]:
```
wk = union.boundary.xy
xs, ys = wk[0].tolist(), wk[1].tolist()
x0, y0 = xs[0], ys[0]
locations = list(zip(xs, ys))
```

[58]:
```
import folium
map_osm = folium.Map(location=[y0, x0])
map_osm.add_child(folium.PolyLine(locations=[(_[1], _[0]) for _ in shape.points],
                                  popup=format_popud(d), color="#000000"))
folium_html_map(map_osm, width="50%")
```

[58]:
```
<pyensae.notebookhelper.folium_helper.folium_html_map.<locals>.CustomFoliumMap
at 0x1cb4e27ab38>
```

1.6.7 Carte finale après fusion des contours

Cette fusion repose sur la fonctionnalité que nous venons de présenter à savoir la fusion de deux contours. Il faut aussi pouvoir associer un contour avec la solution gagnante. Cette solution a pour format `{ circonscription : [ liste des bureaux ] }`. On rappelle les identifiants choisis :

- identifiant cironscription : **DDCCC#**, `D` pour code département, `C` pour code circonscription
- identifiant bureau de vote : **DDMMMAABBB#**, `D` pour code département, `M` pour code commune, `C` pour code commune, `A` pour code canton, le code du bureau est laissé à `***` si les données géolocalisées donne le même lieu pour plusieurs bureaux de vote.

Example avec le premier bureau :

[59]:
```
d = {k[0]:v for k,v in zip(rshp.fields[1:], records[0])}
d
```

[59]:
```
{'BUREAU': '01001',
 'CODE': '01001',
 'NOM': "L'Abergement-Clmenciat",
 'CODEARRT': '012',
 'CODEDEP': '01',
 'CODEREG': '82',
 'CODECANT': '10',
 'CANTON': 'CHATILLON-SUR-CHALARONNE',
 'CIRCO': '04'}
```

fonction 1 : créer un dictionnaire avec les contours des bureaux de vote Les shapes sont dans un tableau indicés par des entiers. Il sera plus simple de les indicés par leur identidiant.

[60]:
```
shape_bureau = {}
for rec, shape in zip(records, shapes):
    d = {k[0]:v for k,v in zip(rshp.fields[1:], rec)}
    idbureau = shape_idbureau(d)
    shape_bureau[idbureau] = (d, shape)
    d["IDB"] = idbureau
```

[61]:
```
list(sorted(shape_bureau.items()))[2006:2008]
```

[61]:
```
[('0506130013#',
  ({'BUREAU': '05061_013',
    'CODE': '05061',
    'NOM': 'Gap',
    'CODEARRT': '052',
    'CODEDEP': '05',
    'CODEREG': '93',
    'CODECANT': '30',
    'CANTON': 'Gap-Sud-Ouest',
    'CIRCO': '01',
    'IDB': '0506130013#'},
   <shapefile.Shape at 0x1cb026c0da0>)),
 ('0506220***#',
  ({'BUREAU': '05062',
    'CODE': '05062',
    'NOM': 'Le Glaizil',
    'CODEARRT': '052',
    'CODEDEP': '05',
    'CODEREG': '93',
    'CODECANT': '20',
    'CANTON': 'SAINT-FIRMIN',
    'CIRCO': '02',
    'IDB': '0506220***#'},
   <shapefile.Shape at 0x1cb108590b8>))]
```

fonction 2 : transformer les idbureau dans la base initiale Rappel : nous n'avons pas la localisation de tous les bureaux de vote. Certains ont été agrégés. On construit alors un nouveau identifiant `idbureau2` pour les bureaux de votes agrégés.

[62]:
```
def new_idbureau(r):
    if r in shape_bureau:
        return r
    else:
        return r[:-4] + "***#"

t1t2noz = t1t2noz.copy()
t1t2noz["idbureau2"] = t1t2noz["idbureau"].apply(lambda r: new_idbureau(r))
```

[63]:
```
t1t2noz[list(_ for _ in t1t2noz.columns if "candidat" not in _ and ("Code" in _ or␣
 ↪"id" in _ or "N°" in _))].head(n=2)
```

[63]:

	Code de la commune	Code département	N° de bureau de vote	N° de canton	\
47442	4	06	0101	1	
47443	4	06	0101	1	

```
       N° de circonscription Lg  N° tour     idbureau  idcirc    idbureau2
47442                         7        1  0600401101#  06007#  0600401***#
47443                         7        1  0600401101#  06007#  0600401***#
```

On vérifie qu'on ne s'est pas trompé et que certains identifiants ont été retrouvés.

[64]:
```
t1t2noz["idb="] = t1t2noz["idbureau"] == t1t2noz["idbureau2"]
t1t2noz["idbgeo"] = t1t2noz["idbureau2"].apply(lambda r: r in shape_bureau)
t1t2noz[["idb=", "idbgeo", "Nombre de voix du candidat"]].groupby(["idb=", "idbgeo"]).
↪sum()
```

[64]:
```
                Nombre de voix du candidat
idb=  idbgeo
False False                        2866063
      True                        15351963
True  True                         5478231
```

Ce sont près de 2.8 millions de voix que nous n'arrivons pas à localiser. Nous ne pourrons pas les changer de circonscriptions. Regardons un identifiant du bureau de vote non localisé.

[65]:
```
t1t2noz[~t1t2noz["idb="] & ~t1t2noz["idbgeo"]].head(n=2)
```

[65]:
```
       Code de la commune Code département Code nuance du candidat  Exprimés  \
47658                   4               06                     EXG       327
47659                   4               06                      FN       327

       Inscrits Nom de la commune Nom du candidat  Nombre de voix du candidat  \
47658       598           Antibes          PETARD                          4
47659       598           Antibes            VIOT                         68

      N° de bureau de vote  N° de canton  …  N° de dépôt du candidat  \
47658                0201            47  …                        8
47659                0201            47  …                       24

       N° tour  Prénom du candidat Votants   elu     idbureau  idcirc  \
47658        1           Christian     334  True  0600447201#  06007#
47659        1            Mathilde     334  True  0600447201#  06007#

         idbureau2   idb=  idbgeo
47658  0600447***#  False   False
47659  0600447***#  False   False

[2 rows x 21 columns]
```

[66]:
```
list(set((t1t2noz[~t1t2noz["idb="] & ~t1t2noz["idbgeo"]])["Nom de la commune"]))[:5]
```

[66]:
```
['Coti-Chiavari', 'Fozzano', 'Vitry-le-François', 'Perelli', 'Vertou']
```

[67]:
```
t1t2noz[~t1t2noz["idb="] & ~t1t2noz["idbgeo"] & (t1t2noz["Nom de la commune"] ==␣
↪"Avelin")].head(n=10).T
```

[67]:
```
                                73887        73888
Code de la commune                 34           34
Code département                   59           59
```

Code nuance du candidat	SOC	UMP
Exprimés	255	255
Inscrits	438	438
Nom de la commune	Avelin	Avelin
Nom du candidat	DEFFONTAINE	LAZARO
Nombre de voix du candidat	116	139
N° de bureau de vote	0003	0003
N° de canton	48	48
N° de circonscription Lg	6	6
N° de dépôt du candidat	91	145
N° tour	2	2
Prénom du candidat	Angélique	Thierry
Votants	271	271
elu	NaN	NaN
idbureau	5903448003#	5903448003#
idcirc	59006#	59006#
idbureau2	5903448***#	5903448***#
idb=	False	False
idbgeo	False	False

[68]:
```
list((k,v[0]) for k, v in shape_bureau.items() if v[0]["NOM"] == "Avelin")
```

[68]:
```
[('5903448001#',
  {'BUREAU': '59034_001',
   'CODE': '59034',
   'NOM': 'Avelin',
   'CODEARRT': '595',
   'CODEDEP': '59',
   'CODEREG': '31',
   'CODECANT': '48',
   'CANTON': 'Pont--Marcq',
   'CIRCO': '06',
   'IDB': '5903448001#'}),
 ('5903448002#',
  {'BUREAU': '59034_002',
   'CODE': '59034',
   'NOM': 'Avelin',
   'CODEARRT': '595',
   'CODEDEP': '59',
   'CODEREG': '31',
   'CODECANT': '48',
   'CANTON': 'Pont--Marcq',
   'CIRCO': '06',
   'IDB': '5903448002#'})]
```

Visiblement les bureaux de vote sont différents dans les deux bases pour la ville d'Avelin. Où est-ce ? D'après l'internaute, il y a 3 bureaux de vote. On peut supposer que ces données viennent d'une mise à jour de la définition de bureaux de vote. D'après l'INSEE, la population croît à Avelin. Il n'est pas improbable qu'un nouveau bureau de vote ait été créé.

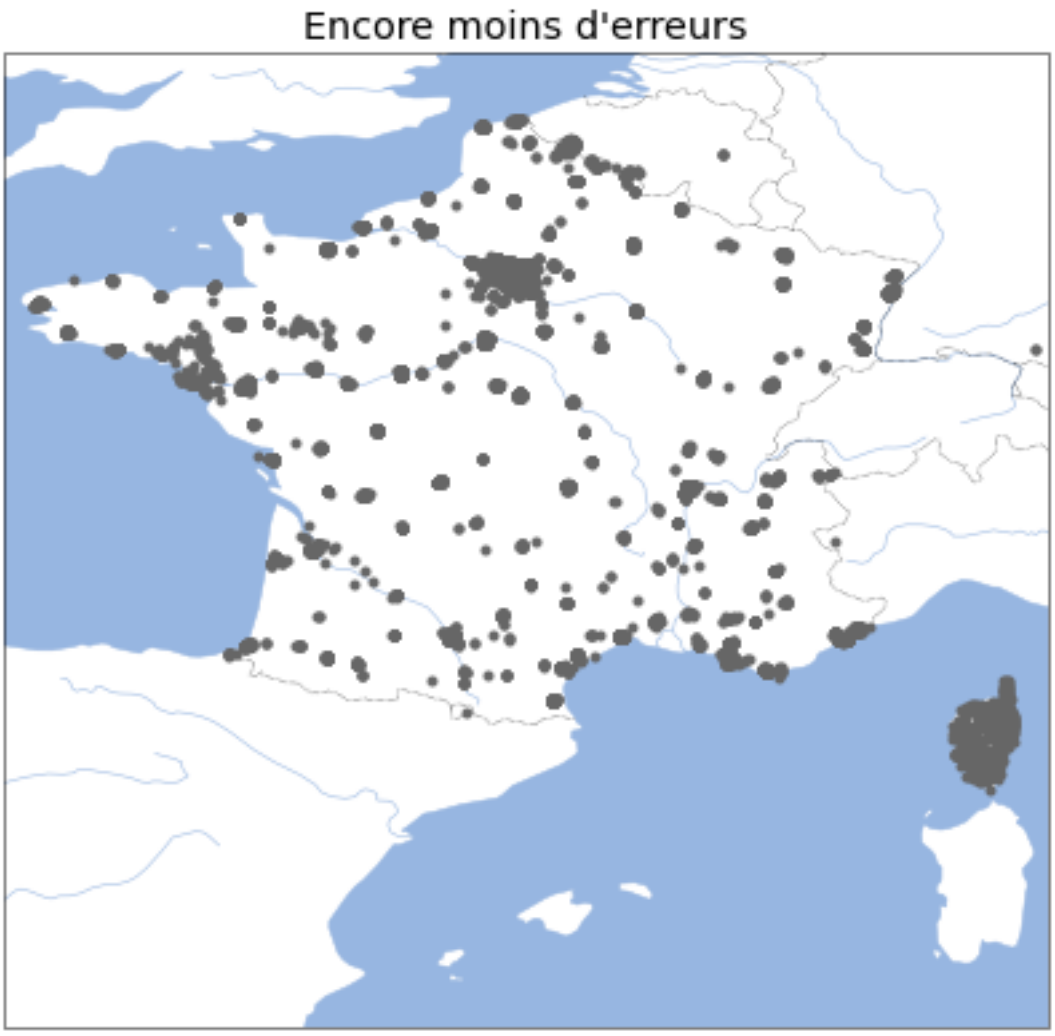
[69]:
```
locs = list((k,v[0],v[1]) for k, v in shape_bureau.items() if v[0]["NOM"] == "Avelin")
x0, y0 = locs[0][2].points[0]
map_osm = folium.Map(location=[y0, x0])
```

```
map_osm.add_child(folium.PolyLine(locations=[(_[1], _[0]) for _ in locs[0][2].points],␣
  ↪color="#FF0000"))
map_osm.add_child(folium.PolyLine(locations=[(_[1], _[0]) for _ in locs[1][2].points],␣
  ↪color="#0000FF"))
folium_html_map(map_osm, width="50%")
```

[69]:
```
<pyensae.notebookhelper.folium_helper.folium_html_map.<locals>.CustomFoliumMap
at 0x1cb4c8694a8>
```

Regardons les bureaux localisés mais non répertoriés dans la base de vote.

[70]:
```
ax = carte_france()
lons = bureau_geo["longitude"]
lats = bureau_geo["latitude"]
ax.plot(lons, lats, ".", color=(0.4, 0.4, 0.4))
ax.set_title("Encore moins d'erreurs");

ok_bureau = set(t1t2noz["idbureau2"])

lons = []
lats = []
for k, v in shape_bureau.items():
    if k in ok_bureau:
        # les bureaux sans voix
        continue
    x1, y1, x2, y2 = v[1].bbox
    x = (x1+x2) / 2
    y = (y1+y2) / 2
    lons.append(x)
    lats.append(y)

ax.plot(lons, lats, ".", color=(0.4, 0.4, 0.4));
```



récupérer les 2.8 M de voix non localisées
Pour les récupérer, nous allons agréger les bureaux de vote de la même commune et cantons en supposant que les erreurs commises ne seront pas trop grandes. Cette fois-ci c'est la variable `shape_bureau` qu'il faut modifier en fusionnant les bureaux pour lesquels nous n'avons pas de voix. Le tableau suivant résume les différents traitements que nous devons faire. L'étape 1 a déjà été faite. Il reste les étapes 2 et 3.

étape	voix (t1t2noz)	localisation (shape_bureau)
0	agrégation par bureau de vote	agrégation par bureau de vote et aussi par commune (***)
1	on corrige les identifiants de bureau non localisés en supposant qu'ils sont agrégés par commune (ajout de ***)	
2		Les bureaux localisés mais sans voix associées ont disparu suite à un redécoupage. On les agrège au niveau de la commune (ajout de ***)

étape	voix (t1t2noz)	localisation (shape_bureau)
3	On agrège au niveau de la commune les bureaux agrégés localement par l'étape 2 (c'est-à-dire qu'on réapplique l'étape 1	

On met à jour les identifiants des contours des bureaux :

[71]:
```
idbureau_voix = set(t1t2noz["idbureau2"])
shape_bureau_list = {}
for k, v in shape_bureau.items():
    if k not in idbureau_voix:
        # on enlève l'indice du bureau
        idb = k[:-4] + "***#"
    else:
        idb = k
    if idb not in shape_bureau_list:
        shape_bureau_list[idb] = []
    shape_bureau_list[idb].append(v)
len(shape_bureau), len(shape_bureau_list)
```

[71]:
```
(50521, 46762)
```

On fusionne les contours et on convertit les autres pour obtenir Polygon ou MultiPolygon. Les MultiPolygon surviennent lorsque des bureaux de vote n'ont pas de bords en commun. Les contours sont décrits plus en détail sur wikipédia : shapefile. Il reste quelques incohérences dans les informations associées à chaque forme. Le commentaire précise comment en trouver.

[72]:
```
import copy
from shapely.geometry import MultiPolygon

def contour2Polygon(obj):
    if obj.shapeType != 5:
        raise Exception("Polygone attendu :\n{0}".format(obj.__dict__))
    points = []
    last = None
    for x, y in obj.points:
        pp = Point(x, y)
        if last is not None and last.almost_equals(p):
            continue
        points.append((x, y))
    pol = Polygon(points)
    # simplifie le polygone
    pol = pol.simplify(tolerance=1e-5)
    # lire http://stackoverflow.com/questions/13062334/
    ↪polygon-intersection-error-python-shapely
    # corrige les polygones qui se croisent.
    return pol.buffer(0)

def fusion_contours(idb, contours):
    d0 = contours[0][0]
    d = d0.copy()
    d["BUREAU"] = d["BUREAU"].split("_")[0]
    sh = copy.deepcopy(contours[0][1])
    shapes = []
```

```
    for i, c in enumerate(contours):
        for k, v in c[0].items():
            # enlever CIRCO de la liste pour trouver des incohérences
            if k not in ("BUREAU", "CIRCO", "IDB", "NOM") and d[k] != v:
                raise Exception(
                    "Incohérence:\n{0}\n{1}\nk={2}\nidb={3}\ncheck={4}".format(
                        d0, c[0], k, idb, shape_idbureau(c[0])))
            pol = contour2Polygon(c[1])
            if isinstance(pol, MultiPolygon):
                shapes.extend(pol.geoms)
            else:
                shapes.append(pol)

    multi = MultiPolygon(shapes)
    return d, multi, cascaded_union(multi)


new_shape_bureau = {}
for ic, (k, v) in enumerate(shape_bureau_list.items()):
    if len(new_shape_bureau) % 1000 == 0:
        print(k, len(v), len(new_shape_bureau), "/", len(shape_bureau_list))
    if len(v) == 1:
        if not isinstance(v[0][1], Polygon):
            new_shape_bureau[k] = v[0][0], contour2Polygon(v[0][1])
        else:
            new_shape_bureau[k] = v[0]
        if not isinstance(new_shape_bureau[k][1], (Polygon, MultiPolygon)):
            raise TypeError(type(new_shape_bureau[k][1]))
    else:
        # fusion
        key, multi, poly = fusion_contours(k, v)
        # poly peut être un Polygon ou un MultiPolygon
        # les bureaux ne sont pas toujours voisins
        if not isinstance(poly, (Polygon, MultiPolygon)):
            raise TypeError(type(poly))
        new_shape_bureau[k] = key, poly
```

```
0100110***# 1 0 / 46762
0255823***# 1 1000 / 46762
0506126023# 1 2000 / 46762
0810532030# 1 3000 / 46762
1030907***# 1 4000 / 46762
1300147017# 1 5000 / 46762
1452049***# 1 6000 / 46762
1709808***# 1 7000 / 46762
1921322***# 1 8000 / 46762
2210715***# 1 9000 / 46762
2442815***# 1 10000 / 46762
2623818***# 1 11000 / 46762
2809601***# 1 12000 / 46762
2B159NA***# 1 13000 / 46762
3140025***# 1 14000 / 46762
3315422***# 1 15000 / 46762
3423914***# 1 16000 / 46762
```

```
3712607***# 1 17000 / 46762
3919932***# 1 18000 / 46762
4129328***# 1 19000 / 46762
4415959***# 1 20000 / 46762
4702306***# 1 21000 / 46762
5012925026# 1 22000 / 46762
5145422035# 1 23000 / 46762
5326316***# 1 24000 / 46762
5535317***# 1 25000 / 46762
5746004***# 1 26000 / 46762
5915572003# 1 27000 / 46762
5957442004# 1 28000 / 46762
6102711***# 1 29000 / 46762
6237810***# 1 30000 / 46762
6335929***# 1 31000 / 46762
6524814***# 1 32000 / 46762
6743721***# 1 33000 / 46762
6925646008# 1 34000 / 46762
7121551***# 1 35000 / 46762
7312602***# 1 36000 / 46762
7629821001# 1 37000 / 46762
7717728***# 1 38000 / 46762
7907122***# 1 39000 / 46762
8068037***# 1 40000 / 46762
8312624***# 1 41000 / 46762
8703038***# 1 42000 / 46762
8917229***# 1 43000 / 46762
9200905016# 1 44000 / 46762
9302710005# 1 45000 / 46762
9405544002# 1 46000 / 46762
```

Dernière étape : on corrige à nouveau les identifiants dans la base des votes

[73]:
```
def new_idbureau2(r):
    if r in new_shape_bureau:
        return r
    else:
        return r[:-4] + "***#"

t1t2noz = t1t2noz.copy()
t1t2noz["idbureau3"] = t1t2noz["idbureau2"].apply(lambda r: new_idbureau2(r))
```

[74]:
```
t1t2noz["idb2="] = t1t2noz["idbureau2"] == t1t2noz["idbureau3"]
t1t2noz["idbgeo2"] = t1t2noz["idbureau3"].apply(lambda r: r in new_shape_bureau)
t1t2noz[["idb2=", "idbgeo2", "Nombre de voix du candidat"]].groupby(["idb2=",␣
 ↪"idbgeo2"]).sum()
```

[74]:
```
                Nombre de voix du candidat
idb2= idbgeo2
True  False                        1490020
      True                        22206237
```

On n'a pas changé grand-chose côté base de vote mais le matching avec la base de localisation a été accru. Nous sommes tombés à 1.5 millions de voix non localisées au lieu de 2.8 millions. Regardons quelques lignes :

[75]:
```
t1t2noz[~t1t2noz.idbgeo2].head(n=2)
```

[75]:
```
       Code de la commune Code département Code nuance du candidat  Exprimés  \
47658                   4               06                     EXG       327
47659                   4               06                      FN       327

       Inscrits Nom de la commune Nom du candidat  Nombre de voix du candidat  \
47658       598           Antibes          PETARD                          4
47659       598           Antibes            VIOT                         68

      N° de bureau de vote  N° de canton  …  Votants   elu     idbureau  \
47658                 0201            47  …      334  True  0600447201#
47659                 0201            47  …      334  True  0600447201#

       idcirc    idbureau2   idb= idbgeo    idbureau3 idb2=  idbgeo2
47658  06007#  0600447***#  False  False  0600447***#  True    False
47659  06007#  0600447***#  False  False  0600447***#  True    False

[2 rows x 24 columns]
```

Quelques villes :

[76]:
```
list(sorted(set(t1t2noz[~t1t2noz.idbgeo2]["Nom de la commune"])))[:5]
```

[76]:
```
['Adelans-et-le-Val-de-Bithaine', 'Afa', 'Aghione', 'Aiti', 'Aix-en-Provence']
```

[77]:
```
t1t2noz[~t1t2noz.idbgeo2 & (t1t2noz["Nom de la commune"] == "Afa")].head()
```

[77]:
```
       Code de la commune Code département Code nuance du candidat  Exprimés  \
24490                   1              2A                     DVG       638
24491                   1              2A                     UMP       638
24492                   1              2A                     DVG       701
24493                   1              2A                     UMP       701
24494                   1              2A                     DVG       173

       Inscrits Nom de la commune Nom du candidat  Nombre de voix du candidat  \
24490       952              Afa         RENUCCI                        442
24491       952              Afa      MARCANGELI                        196
24492      1065              Afa         RENUCCI                        482
24493      1065              Afa      MARCANGELI                        219
24494       246              Afa         RENUCCI                        128

      N° de bureau de vote  N° de canton  …  Votants  elu     idbureau  \
24490                 0001            73  …      657  NaN  2A00173001#
24491                 0001            73  …      657  NaN  2A00173001#
24492                 0002            73  …      727  NaN  2A00173002#
24493                 0002            73  …      727  NaN  2A00173002#
24494                 0003            73  …      177  NaN  2A00173003#

       idcirc    idbureau2   idb= idbgeo    idbureau3 idb2=  idbgeo2
24490  2A001#  2A00173***#  False  False  2A00173***#  True    False
24491  2A001#  2A00173***#  False  False  2A00173***#  True    False
24492  2A001#  2A00173***#  False  False  2A00173***#  True    False
24493  2A001#  2A00173***#  False  False  2A00173***#  True    False
```

```
24494  2A001#  2A00173***#  False  False  2A00173***#  True    False

[5 rows x 24 columns]
```

[78]:
```
[(k, v) for k, v in new_shape_bureau.items() if v[0]["NOM"] == "Afa"]
```

[78]:
```
[('2A001NA***#',
  ({'BUREAU': '2A001',
    'CODE': '2A001',
    'NOM': 'Afa',
    'CODEARRT': '2A1',
    'CODEDEP': '2A',
    'CODEREG': '94',
    'CODECANT': 'NA',
    'CANTON': 'NA',
    'CIRCO': '01',
    'IDB': '2A001NA***#'},
   <shapely.geometry.polygon.Polygon at 0x1cb535464a8>))]
```

Le code canton n'est pas renseigné.

[79]:
```
t1t2noz[~t1t2noz.idbgeo2 & (t1t2noz["Nom de la commune"] == "Aix-en-Provence")].
↪sort_values("idbureau")
```

[79]:
```
       Code de la commune Code département Code nuance du candidat  Exprimés  \
16364                   1               13                     UMP       378
16365                   1               13                     SOC       378
16366                   1               13                     UMP       456
16367                   1               13                     SOC       456
16368                   1               13                     UMP       374
16369                   1               13                     SOC       374

       Inscrits Nom de la commune   Nom du candidat  \
16364       700   Aix-en-Provence  JOISSAINS-MASINI
16365       700   Aix-en-Provence              CIOT
16366       761   Aix-en-Provence  JOISSAINS-MASINI
16367       761   Aix-en-Provence              CIOT
16368       629   Aix-en-Provence  JOISSAINS-MASINI
16369       629   Aix-en-Provence              CIOT

       Nombre de voix du candidat N° de bureau de vote  N° de canton  …  \
16364                         128                 0083            47  …
16365                         250                 0083            47  …
16366                         209                 0084            47  …
16367                         247                 0084            47  …
16368                         220                 0085            47  …
16369                         154                 0085            47  …

       Votants  elu     idbureau  idcirc    idbureau2   idb= idbgeo  \
16364      386  NaN  1300147083#  13014#  1300147***#  False  False
16365      386  NaN  1300147083#  13014#  1300147***#  False  False
16366      463  NaN  1300147084#  13014#  1300147***#  False  False
16367      463  NaN  1300147084#  13014#  1300147***#  False  False
16368      379  NaN  1300147085#  13014#  1300147***#  False  False
```

```
16369      379  NaN  1300147085#  13014#  1300147***#  False  False

         idbureau3 idb2=  idbgeo2
16364  1300147***#  True    False
16365  1300147***#  True    False
16366  1300147***#  True    False
16367  1300147***#  True    False
16368  1300147***#  True    False
16369  1300147***#  True    False

[6 rows x 24 columns]
```

[80]:
```
[(k, v) for k, v in new_shape_bureau.items() if v[0]["NOM"] == "Aix-en-Provence" and␣
 ↪v[0]["CODECANT"] == "47"][0]
```

[80]:
```
('1300147001#',
 ({'BUREAU': '13001_001',
   'CODE': '13001',
   'NOM': 'Aix-en-Provence',
   'CODEARRT': '131',
   'CODEDEP': '13',
   'CODEREG': '93',
   'CODECANT': '47',
   'CANTON': 'Aix-en-Provence-Centre',
   'CIRCO': '14',
   'IDB': '1300147001#'},
  <shapely.geometry.polygon.Polygon at 0x1cb4ff8c1d0>))
```

[81]:
```
len(set(t1t2noz[t1t2noz["Nom de la commune"] == "Aix-en-Provence"]["idbureau"]))
```

[81]:
```
87
```

[82]:
```
len([(k, v) for k, v in new_shape_bureau.items() if v[0]["NOM"] == "Aix-en-Provence"])
```

[82]:
```
84
```

Il y avait 3 bureaux de vote de moins en 2007 par rapport à 2012. Pour cette petite ville, il serait possible d'agréger ces bureaux ensemble sans impacter les résultats. Voyons déjà si on peut faire sans.

fonction 3 : dessiner les bureaux avec la couleur des circonscriptions On essaye de retrouver la carte des circonscriptions obtenues plus haut mais sans utiliser la table des contours des circonscriptions. Nous n'avons pas besoin de fusionner les contours des bureaux pour obtenir les contours des circonscriptions, juste de représenter chaque bureau avec la couleur (gagnant, perdant) qui lui est associée. La couleur ne dépend pas de la couleur du bureau ne dépend pas de ses résultats mais de ceux de la circonscription à laquelle il est associé. Pour tracer la carte, on peut soit faire comme expliqué ci-dessus ou fusionner les contours des bureaux pour obtenir ceux des circonscriptions. Il apparaît que la première solution est plus simple et pas plus longue. Comme il faut compter environ 3 minutes pour tracer la carte, nous n'allons pas le faire souvent.

[83]:
```
from itertools import groupby
from shapely.geometry import MultiPolygon

def new_agg_bureau_shape_viz(thewinner, shape_bureau, data_vote, solution=None,
```

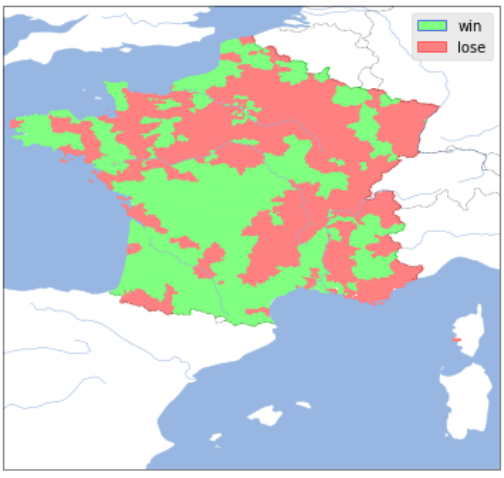
```
                col_circ="idcirc", col_place="idbureau", col_vote="Nombre de voix du candidat",
                col_nuance="Code nuance du candidat", figsize=(14,6), **kwargs):
    """
    Visualise la nuance gagnante dans chaque circonscription.

    @param      thewinner    parti qu'on souhaite influencer
    @param      shape_bureau dictionnaire ``{ idbureau : (information, shapefile)}``
    @param      figsize      dimension du graphiques
    @param      kwargs       options additionnelles

    @param      data_vote    dataframe de type
    @param      solution     dictionnaire ``{ circonscription : liste de bureaux }``,
                             si None considère la solution officielle
    @param      col_circ     colonne contenant la circonscription (si solution = None)
    @param      col_place    colonne contenant l'identifiant du bureaux de votes
    @param      col_vote     colonne contenant les votes
    @param      col_nuance   colonne contenant le parti ou la nuance
    @return                  matrice de resultat, une ligne par circoncription, une colonne par nuance/parti
    """
    # on transforme les dataframes en dictionnaires
    score = agg_circonscription(data_vote, solution=solution,
                col_circ=col_circ, col_place=col_place, col_vote=col_vote,
                col_nuance=col_nuance)
    winner = score[["winner"]].to_dict("index")

    if solution is None:
        # pas de solution, on récupère la configuration existante
        # il ne faut pas oublier de choisir idbureau3
        gr = data_vote[["idcirc", "idbureau3", "Code département"]].groupby(["idcirc", "idbureau3"], as_index=False).count()
        gr = gr[["idcirc", "idbureau3"]].sort_values("idcirc")
        solution = {}
        for k, g in groupby(gr.values, lambda d: d[0]):
            solution[k] = list(_[1] for _ in g)
            if len(solution[k]) == 0:
                raise Exception("group should not be empty\nk={0}\ng={1}".format(k, list(g)))


    fig = plt.figure(figsize=figsize)

    ax1 = fig.add_subplot(1, 2, 1, projection=ccrs.PlateCarree())
    ax1.set_extent([-5, 10, 38, 52])
    ax1.add_feature(cfeature.OCEAN.with_scale('50m'))
    ax1.add_feature(cfeature.RIVERS.with_scale('50m'))
    ax1.add_feature(cfeature.BORDERS.with_scale('50m'), linestyle=':')

    ax2 = fig.add_subplot(1, 2, 2)
    axes = [ax1, ax2]

    # on dessine la distribution des circonscriptions
```

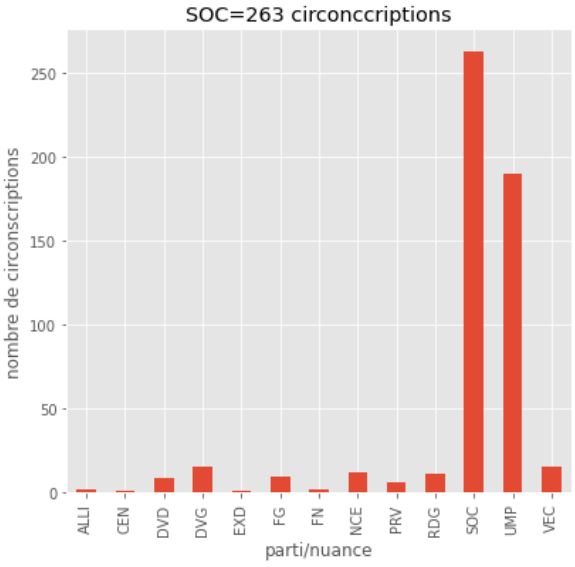
```
count = score[["winner", "nbwinner"]].groupby(["winner"]).count()
count.sort_values("nbwinner", ascending=False)
count.plot(ax=axes[1], kind="bar", legend=False)
axes[1].set_xlabel("parti/nuance")
axes[1].set_ylabel("nombre de circonscriptions")

# on calcule le nombre de places le parti considéré
count = count.reset_index(drop=False)
count["iswin"] = count["winner"] == thewinner
ratio = count[["nbwinner", "iswin"]].groupby("iswin").sum().sort_index()
nbcirc = ratio.iloc[1,0]
axes[1].set_title("{0}={1} circonccriptions".format(thewinner, nbcirc))

def dedup(ps):
    res = []
    for p in ps:
        if p not in res:
            res.append(p)
    return res

polys = []
colors = []

associated = 0
total = 0
for icirc, (idcirc, idbureau) in enumerate(sorted(solution.items())):
    avance = "{0}/{1}".format(icirc, len(solution))
    shapes = []
    for idb in idbureau:
        if idb in shape_bureau:
            obj = shape_bureau[idb][1]
            # les contours ne sont pas toujours des lignes continues,
            # ça peut être plusieurs Polygon ou MultiPolygon
            # et les contours des Line ou MultiLine
            if isinstance(obj, MultiPolygon):
                # MultiPolygon
                for o in obj:
                    try:
                        shapes.append(o.boundary.coords)
                    except:
                        try:
                            for oo in o.boundary.geoms:
                                shapes.append(oo.coords)
                        except Exception as e:
                            raise TypeError(obj.boundary.wkt) from e
            else:
                try:
                    shapes.append(obj.boundary.coords)
                except:
                    try:
                        for o in obj.boundary.geoms:
                            shapes.append(o.coords)
                    except Exception as e:
```

```
                            raise TypeError(obj.boundary.wkt) from e
        associated += len(shapes)
        total += len(idbureau)
        if len(shapes) == 0:
            if len(idbureau) > 3:
                idbureau = idbureau[:3] + ["..."]
            print(avance, "Number of shapes is empty for circonscription={0}␣
↪idbureau={1}".format(idcirc, idbureau))
            continue

        if idcirc in winner:
            win = winner[idcirc]["winner"]
            color = (0.5, 1.0, 0.5) if win == thewinner else (1.0, 0.5, 0.5)
        else:
            color = "black"

        # on dessine tous les bureaux de la même couleur
        for shape in shapes:

            if len(shape) < 3:
                continue
            poly = Polygon(shape)
            polys.append(poly)
            colors.append(color)

    data = geopandas.GeoDataFrame(dict(geometry=polys, colors=colors))
    geopandas.plotting.plot_polygon_collection(axes[0], data['geometry'],␣
↪facecolor=data['colors'],
                                               values=None, edgecolor=data['colors'])

    legend_elements = [Patch(facecolor=(0.5, 1.0, 0.5), edgecolor='b', label='win'),
                       Patch(facecolor=(1.0, 0.5, 0.5), edgecolor='r', label='lose')]
    axes[0].legend(handles=legend_elements, loc='upper right')
    return fig, axes

new_agg_bureau_shape_viz("SOC", new_shape_bureau, t1t2noz);
```

```
123/539 Number of shapes is empty for circonscription=2B001#
idbureau=['2B25726***#', '2B23929***#', '2B23342***#', '…']
124/539 Number of shapes is empty for circonscription=2B002#
idbureau=['2B24639***#', '2B24559***#', '2B24410***#', '…']
349/539 Number of shapes is empty for circonscription=69001#
idbureau=['6912355***#', '6912336***#', '6912315***#', '…']
350/539 Number of shapes is empty for circonscription=69002#
idbureau=['6912311***#', '6912313***#', '6912314***#', '…']
351/539 Number of shapes is empty for circonscription=69003#
idbureau=['6912319***#', '6912320***#', '6912322***#', '…']
352/539 Number of shapes is empty for circonscription=69004#
idbureau=['6912354***#', '6912321***#', '6912322***#', '…']
385/539 Number of shapes is empty for circonscription=75001#
idbureau=['7505623***#', '7505615***#', '7505616***#', '…']
386/539 Number of shapes is empty for circonscription=75002#
idbureau=['7505621***#', '7505619***#', '7505620***#']
```

```
387/539 Number of shapes is empty for circonscription=75003#
idbureau=['7505631***#', '7505632***#']
388/539 Number of shapes is empty for circonscription=75004#
idbureau=['7505630***#', '7505631***#']
389/539 Number of shapes is empty for circonscription=75005#
idbureau=['7505617***#', '7505624***#']
390/539 Number of shapes is empty for circonscription=75006#
idbureau=['7505634***#', '7505625***#']
391/539 Number of shapes is empty for circonscription=75007#
idbureau=['7505618***#', '7505625***#', '7505626***#']
392/539 Number of shapes is empty for circonscription=75008#
idbureau=['7505626***#', '7505634***#']
393/539 Number of shapes is empty for circonscription=75009#
idbureau=['7505627***#']
394/539 Number of shapes is empty for circonscription=75010#
idbureau=['7505628***#', '7505627***#']
395/539 Number of shapes is empty for circonscription=75011#
idbureau=['7505628***#', '7505620***#']
396/539 Number of shapes is empty for circonscription=75012#
idbureau=['7505621***#', '7505629***#']
397/539 Number of shapes is empty for circonscription=75013#
idbureau=['7505629***#']
398/539 Number of shapes is empty for circonscription=75014#
idbureau=['7505630***#']
399/539 Number of shapes is empty for circonscription=75015#
idbureau=['7505634***#']
400/539 Number of shapes is empty for circonscription=75016#
idbureau=['7505633***#']
401/539 Number of shapes is empty for circonscription=75017#
idbureau=['7505632***#', '7505633***#']
402/539 Number of shapes is empty for circonscription=75018#
idbureau=['7505632***#', '7505623***#']
```





En jaune, on trouve le nombre de bureaux de votes effectivement associés à une circonscription.

1.7 Calcul d'une nouvelle affectation

La fonction *agg_circonscription* définie au début du notebook permet de calcul le score d'une association circonscription - bureaux) avec le code suivant :

1.7.1 Calcul d'un score

[84]:
```
score = agg_circonscription(t1t2noz)
count = score[["winner", "nbwinner"]].groupby(["winner"]).count()
count.sort_values("nbwinner", ascending=False).head(n=3)
```

[84]:
```
Code nuance du candidat  nbwinner
winner
SOC                          263
UMP                          190
DVG                           16
```

A partir de cela, on fabrique la fonction *scope_circonscription* qui retourne le score du parti qu'on souhaite faire gagner.

[85]:
```
def score_circonscription(data_vote, thewinner, solution=None, col_circ="idcirc",
                          col_place="idbureau", col_vote="Nombre de voix du candidat",
                          col_nuance="Code nuance du candidat"):
    """
    Calcule le nombre de députés pour un parti donné.

    @param      data_vote    dataframe pour les voix
    @param      thewinner    le parti considéré
    @param      solution     dictionnaire ``{ circonscription : liste de bureaux }``,
                             si None, la fonction considère la solution officielle
    @param      col_circ     colonne contenant la circonscription (si solution = None)
    @param      col_place    colonne contenant l'identifiant du bureaux de votes
    @param      col_vote     colonne contenant les votes
    @param      col_nuance   colonne contenant le parti ou la nuance
    @return                  matrice de résultats, une ligne par circoncription, une␣
 ↪colonne par nuance/parti
    """
    score = agg_circonscription(data_vote, solution=solution, col_circ=col_circ,
                          col_place=col_place, col_vote=col_vote,␣
 ↪col_nuance=col_nuance)
    count = score[["winner", "nbwinner"]].groupby(["winner"], as_index=False).count()
    fcount = count[count["winner"] == thewinner]
    if len(fcount) == 0:
        print("Unable to find '{0}' in '{1}'".format(thewinner, set(fcount["winner"])))
        return 0
    return fcount.reset_index().loc[0,"nbwinner"]

score_circonscription(t1t2noz, "SOC")
```

[85]:
```
263
```

On vérifie que le résultat est le même avec la colonne *idcirc*. L'objectif est de créer une nouvelle colonne dans ce dataframe qui précisera la nouvelle affectation de chaque bureau aux nouvelles circonscription.

[86]:
```
score_circonscription(t1t2noz, "SOC", col_circ="idcirc")
```

[86]:
```
263
```

L'inconvénient de cette métrique est qu'elle est entière. Il est très probable qu'un changement de circonscription pour un bureau ne modifie pas la métrique. C'est problématique car on ne sait pas si un petit changement va dans le bon sens. Nous allons prendre le plus petit département pour lequel nous savons localiser toutes les voix et qui contient au moins 3 circonscriptions. On compte les voix non localisées comme suit :

[87]:
```
miss = t1t2noz[["Code département", "idbgeo2",
         "Nombre de voix du candidat"]].groupby(["Code département",
         "idbgeo2"], as_index=False).sum()
piv = miss.pivot("Code département", "idbgeo2", "Nombre de voix du candidat")
piv.columns = ["Voix non localisées", "Voix localisées"]
piv[piv["Voix non localisées"].isnull()].sort_values("Voix localisées").head(n=8)
```

[87]:
```
                  Voix non localisées  Voix localisées
Code département
48                                NaN          39864.0
90                                NaN          53920.0
23                                NaN          61621.0
32                                NaN          87165.0
18                                NaN         107272.0
39                                NaN         113024.0
12                                NaN         137507.0
88                                NaN         165159.0
```

1.7.2 Essai sur un département simple

Puis, on essaye les premiers départements jusqu'à trouver le numéro 39 (Jura) :

[88]:
```
choix = "39"
```

[89]:
```
t1t2noz[t1t2noz["Code département"] == choix].head(n=2)
```

[89]:
```
       Code de la commune Code département Code nuance du candidat  Exprimés  \
50383                   1               39                     UMP       321
50384                   1               39                     SOC       321

       Inscrits    Nom de la commune Nom du candidat  \
50383       603  Abergement-la-Ronce         SERMIER
50384       603  Abergement-la-Ronce         LAROCHE

       Nombre de voix du candidat N° de bureau de vote  N° de canton  …  \
50383                         175                0001            33  …
50384                         146                0001            33  …

       Votants  elu     idbureau  idcirc    idbureau2   idb= idbgeo  \
50383      335  NaN  3900133001#  39003#  3900133***#  False   True
50384      335  NaN  3900133001#  39003#  3900133***#  False   True

         idbureau3 idb2=  idbgeo2
50383  3900133***#  True     True
50384  3900133***#  True     True

[2 rows x 24 columns]
```

[90]:
```
agg_circonscription(t1t2noz.loc[t1t2noz["Code département"] == choix])
```

[90]:
```
Code nuance du candidat    SOC    UMP winner  nbwinner  total
idcirc
39001#                   19193  20912    UMP      20912  40105
39002#                   14060  16915    UMP      16915  30975
39003#                   19641  22303    UMP      22303  41944
```

[91]:
```
resultat_par_bureau = agg_circonscription(t1t2noz.loc[t1t2noz["Code département"] ==␣
 ↪choix],
                                          col_circ="idbureau3")
resultat_par_bureau.head()
```

[91]:
```
Code nuance du candidat  SOC  UMP winner  nbwinner  total
idbureau3
3900133***#              146  175    UMP       175    321
3900201***#               13    9    SOC        13     22
3900323***#               11   16    UMP        16     27
3900429***#                9   29    UMP        29     38
3900629***#              122   93    SOC       122    215
```

[92]:
```
bex = resultat_par_bureau.reset_index(drop=False).to_dict("records")
bex[:1]
```

[92]:
```
[{'idbureau3': '3900133***#',
  'SOC': 146,
  'UMP': 175,
  'winner': 'UMP',
  'nbwinner': 175,
  'total': 321}]
```

[93]:
```
gr = t1t2noz[t1t2noz["Code département"] ==
             choix][["idcirc", "idbureau3", "Code département"]].groupby(
                 ["idcirc", "idbureau3"], as_index=False).count()
gr = gr[["idcirc", "idbureau3"]].sort_values("idcirc")
asso = {d["idbureau3"]: d["idcirc"] for d in gr.to_dict("records")}
list(asso.items())[:5]
```

[93]:
```
[('3900323***#', '39001#'),
 ('3935423***#', '39001#'),
 ('3936215***#', '39001#'),
 ('3936327***#', '39001#'),
 ('3937521***#', '39001#')]
```

Pour représenter les bureaux, on utilise encore folium et les GeoJSON.

[94]:
```
def iterate_contour(loc):
    try:
        yield loc.boundary.coords
    except Exception as e:
        for mp in loc.boundary.geoms:
            try:
                yield mp.coords
            except Exception as e:
```

```
                raise TypeError(mp.wkt) from e


def create_geojson(loc):
    entities = []
    for points in iterate_contour(loc):
        xy = [ [_[0], _[1]] for _ in points]
        entity = {
            "type": "Feature",
            "geometry": {
                "type": "Polygon",
                "coordinates": [xy]
            },
        }
        entities.append(entity)
    geojson = {"type": "FeatureCollection", "features": entities}
    return geojson
```

[95]:
```
import json

def carte_interactive(bex, asso, new_shape_bureau, colors=None,
                      flag_bureau=None, choix=None):
    """
    Parameters
    ----------

    bex: liste de dictionnaires
         ``[{'SOC': 146, 'UMP': 175, 'idbureau3': '3900133***#',
           'nbwinner': 175, 'total': 321, 'winner': 'UMP'}]``

    asso: dictionnaire { bureau: circonscription }

    new_shape_bureau: dictionnaire contenant les contours,
        clé: ``'1302808014#'``,
        valeur: ``tuple ({'BUREAU': '13028_014', 'CANTON': 'La Ciotat', 'CIRCO': '09',␣
 ↪'CODE': '13028',
        'CODEARRT': '133', 'CODECANT': '08', 'CODEDEP': '13', 'CODEREG': '93', 'IDB':␣
 ↪'1302808014#',
        'NOM': 'La Ciotat'}, <shapely.geometry.polygon.Polygon at 0xa2ab25b208>)

    colors: dictionnaire ``{ circonscription: couleur }``

    flag_bureau: ensemble de bureaux pour lesquels il faut afficher un drapeau ou None␣
 ↪pour tous

    Returns
    -------

    Carte folium
    """
    if colors is None:
        if choix is None:
            raise ValueError("choix must be specified")
```

```
        colors = {choix + '001#':'#FF0000', choix + '002#':'#00FF00', choix + '003#':
↪'#0000FF'}
    map_osm = None
    for bureau in bex:
        winner = bureau["winner"]
        circ = asso[bureau["idbureau3"]]
        loc = new_shape_bureau[bureau["idbureau3"]][1]
        color = colors[circ]
        geo = create_geojson(loc)
        geo_str = json.dumps(geo)
        if map_osm is None:
            print(bureau)
            x0, y0 = geo["features"][0]["geometry"]["coordinates"][0][0]
            map_osm = folium.Map(location=[y0, x0])
        map_osm.choropleth(geo_data=geo_str, fill_color=color, fill_opacity=0.3)

        if flag_bureau is None or bureau["idbureau3"] in flag_bureau:
            mx = [_[0] for _ in geo["features"][0]["geometry"]["coordinates"][0]]
            my = [_[1] for _ in geo["features"][0]["geometry"]["coordinates"][0]]
            mx = sum(mx) / len(mx)
            my = sum(my) / len(my)
            coul = 'green' if winner == "SOC" else 'black'
            map_osm.add_child(folium.Marker(location=(my,mx), icon=folium.
↪Icon(color=coul, icon="circle")))
    return map_osm
```

[96]:
```
map_osm = carte_interactive(bex, asso, new_shape_bureau, choix=choix)
folium_html_map(map_osm, width="70%")
```

```
{'idbureau3': '3900133***#', 'SOC': 146, 'UMP': 175, 'winner': 'UMP',
'nbwinner': 175, 'total': 321}
```

```
c:\python372_x64\lib\site-packages\folium\folium.py:426: FutureWarning: The
choropleth  method has been deprecated. Instead use the new Choropleth class,
which has the same arguments. See the example notebook 'GeoJSON_and_choropleth'
for how to do this.
  FutureWarning
```

[96]:
```
<pyensae.notebookhelper.folium_helper.folium_html_map.<locals>.CustomFoliumMap
at 0x1cb53047cc0>
```

On peut changer la forme des icônes. Maintenant, comment procède-t-on pour changer quelques bureaux de circonscriptions ?

1.7.3 Trouver les bureaux sur les frontières

On s'intéresse aux frontières car il est impossible changer un bureau de vote de circonscription en plein milieu de celle-ci. Il faut que les bureaux de votes d'une même circoncscription soient voisins ou en langage mathématiques forment une ensemble connexe : il doit être possible de passer d'un bout à l'autre de la circonscription sans avoir besoin d'en traverser une autre.

[97]:
```
score_circonscription(t1t2noz, 'SOC')
```

[97]:
```
263
```

[98]:
```
score_circonscription(t1t2noz[t1t2noz["Code département"] == '39'], "SOC")
```

```
Unable to find 'SOC' in 'set()'
```

[98]: 0

[99]:
```
res = []
for choix in set(t1t2noz["Code département"]):
    res.append(dict(choix=choix,
                    score=score_circonscription(t1t2noz[t1t2noz["Code département"] ==␣
 ↪choix], "SOC")))
df = pandas.DataFrame(res)
df.head()
```

```
Unable to find 'SOC' in 'set()'
Unable to find 'SOC' in 'set()'
Unable to find 'SOC' in 'set()'
Unable to find 'SOC' in 'set()'
Unable to find 'SOC' in 'set()'
Unable to find 'SOC' in 'set()'
Unable to find 'SOC' in 'set()'
Unable to find 'SOC' in 'set()'
Unable to find 'SOC' in 'set()'
Unable to find 'SOC' in 'set()'
Unable to find 'SOC' in 'set()'
Unable to find 'SOC' in 'set()'
Unable to find 'SOC' in 'set()'
Unable to find 'SOC' in 'set()'
Unable to find 'SOC' in 'set()'
```

[99]:

	choix	score
0	93	9
1	04	2
2	95	5
3	90	0
4	27	2

On extrait la liste ds bureaux pour les trois circonscriptions et on utilise la fonction distance du module *shapely* pour calculer les distances pour les contours de bureaux dans des circonscriptions existantes. On a également besoin de garder les deux circonscriptions qui contiennent les deux bureaux voisins : on enlève un bureau à une circonscription pour l'ajouter à sa voisine.

[100]:
```
def distance_contour_bureau(bureaux, association):
    distance = {}
    for b1, v1 in bureaux.items():
        for b2, v2 in bureaux.items():
            circ1 = association[b1]
            circ2 = association[b2]
            if circ1 != circ2:
                dist = v1[1].distance(v2[1])
                distance[b1, b2] = (dist, circ1, circ2)
    return distance
```

```
bureau_39 = {k:v for k,v in new_shape_bureau.items() if k in asso}
dist_39 = distance_contour_bureau(bureau_39, asso)
list(dist_39.items())[:2]
```

[100]:
```
[(('3900133***#', '3900323***#'), (0.3121574297008682, '39003#', '39001#')),
 (('3900133***#', '3900721***#'), (0.4844400080127889, '39003#', '39001#'))]
```

On s'intéresse aux distances nulles : des voisins qui ont un sommet en commun et qui sont de circonscriptions différenes.

[101]:
```
import pandas
df = pandas.DataFrame(dict(dist39=list(dist_39.values())))
df[df.dist39==0].shape
```

[101]:
```
(0, 1)
```

On les dessine.

[102]:
```
subset = [k for k, v in dist_39.items() if v[0] == 0]
subset = set([_[0] for _ in subset] + [_[1] for _ in subset])
map_osm = carte_interactive(bex, asso, new_shape_bureau, flag_bureau=subset,␣
  ↪choix='39')
folium_html_map(map_osm, width="70%")
```

```
{'idbureau3': '3900133***#', 'SOC': 146, 'UMP': 175, 'winner': 'UMP',
'nbwinner': 175, 'total': 321}

c:\python372_x64\lib\site-packages\folium\folium.py:426: FutureWarning: The
choropleth  method has been deprecated. Instead use the new Choropleth class,
which has the same arguments. See the example notebook 'GeoJSON_and_choropleth'
for how to do this.
  FutureWarning
```

[102]:
```
<pyensae.notebookhelper.folium_helper.folium_html_map.<locals>.CustomFoliumMap
at 0x1cb5b014160>
```

Ca marche assez bien excepté quelques exceptions autour de grands contours qui n'ont qu'une petite partie commune avec la frontière. Peut-on changer les résultats des élections avec ces bureaux de vote ? Pour rappel :

[103]:
```
choix = '39'
agg_circonscription(t1t2noz.loc[t1t2noz["Code département"] == choix])
```

[103]:

Code nuance du candidat	SOC	UMP	winner	nbwinner	total
idcirc					
39001#	19193	20912	UMP	20912	40105
39002#	14060	16915	UMP	16915	30975
39003#	19641	22303	UMP	22303	41944

On introduit une cironscription temporaire : la frontière.

[104]:
```
res39 = t1t2noz.loc[t1t2noz["Code département"] == choix].copy()
res39["newidcirc"] = res39.apply(lambda row: "frontière" if row["idbureau3"] in subset␣
  ↪else row["idcirc"], axis=1)
agg_circonscription(res39, col_circ="newidcirc")
```

[104]:
Code nuance du candidat	SOC	UMP	winner	nbwinner	total
newidcirc					
39001#	16958	18324	UMP	18324	35282
39002#	12850	15132	UMP	15132	27982
39003#	17690	20008	UMP	20008	37698
frontière	5396	6666	UMP	6666	12062

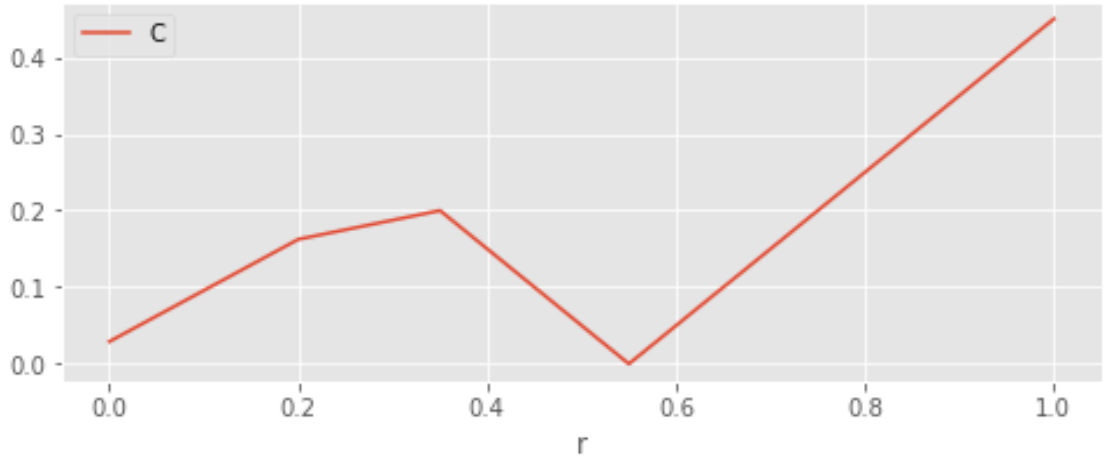
1.7.4 Inventer une fonction de score plus précise

156 bureaux possibles. Ca fait beaucoup. On souhaite trouver un moyen de trier les bureaux par ordre d'intérêt. On souhaite changer le score d'une circonscription. La première circonscription est la plus intéressante car l'écart est le plus faible. Les circonscriptions où l'adversaire a beaucoup de voix est aussi une configuration intéressante. Si r est la proportion de voix associées au parti qu'on souhaite favoriser, on cherche à constuire une fonction de coût $C(r)$ qui vérifie :

- $C(r)$ est faible si $r < 0.4$: impossible de battre l'adversaire
- $C(r)$ est fort si $0.4 < r < 0.51$: la zone où on peut agir et on ne veut pas voir cette configuration
- $C(r)$ est très faible si $0.51 < r < 0.6$: le cas inverse, le parti favorisé gagne la circonscription avec peu de marge
- $C(r)$ est fort is $r > 0.6$, le parti favorisé gagne avec trop de marge, ses voix pourraient être mieux utilisées ailleurs

Il faudrait aussi tenir compte du poids relatif à chaque circonscription dans le département afin d'éviter d'avoir trop d'écart. On s'en passera pour cet exercice en supposant que les changements à la frontière ne vont trop malmener cette distribution. On construit grossièrement la fonction suivante. Les seuils et la forme sont choisis sans réelle étude. Il faudrait aussi se pencher sur la distributions de r et sur les intervalles de confiances obtenus en appliquant un bootstrap : on calcule un grand nombre de fois le ratio r à partir de tirages aléatoires du bureaux au sein d'une même circonscription.
Plus formellement, on cherche à calculer $\mathbb{P}(p = N|r, r')$ qui est la probabilité de gagner pour le parti N sachant le proportion de voix r dans la circonscription et r' la proportion chez sa voisine. On cherche à construire une fonction de coût qui ressemble à quelque chose comme : $\alpha \int_r \mathbb{P}(p = N|r, r')dr + \beta \int_{r'} \mathbb{P}(p = N|r, r')dr'$.

[105]:
```
def fonction_cout(r):
    if r >= 0.55:
        return abs(r-0.55)
    elif r >= 0.4:
        return abs(r-0.55)
    elif r >= 0.35:
        return fonction_cout(0.4) + abs(r-0.4)
    elif r >= 0.2:
        return fonction_cout(0.35) - abs(r - 0.35) / 4
    else:
        return fonction_cout(0.2) - abs(r - 0.2) / 1.5

rx = [i/100.0 for i in range(0, 101)]
Cy =[fonction_cout(r) for r in rx]

import pandas
df = pandas.DataFrame(dict(r=rx, C=Cy))
df.plot(x="r", y="C", figsize=(8,3));
```



Il faut minimiser ce coût pour chaque circonscription.

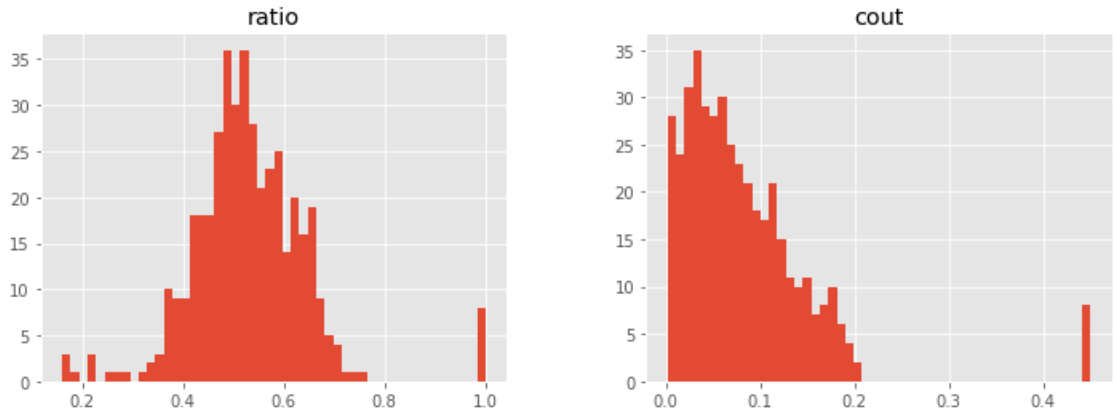
[106]:
```
agg = agg_circonscription(res39, col_circ="newidcirc")
agg["ratio"] = agg["SOC"] / agg["total"]
agg["cout"] = agg["ratio"].apply(fonction_cout)
agg
```

[106]:
```
Code nuance du candidat    SOC    UMP winner  nbwinner  total     ratio  \
newidcirc
39001#                   16958  18324    UMP     18324  35282  0.480642
39002#                   12850  15132    UMP     15132  27982  0.459224
39003#                   17690  20008    UMP     20008  37698  0.469256
frontière                 5396   6666    UMP      6666  12062  0.447355

Code nuance du candidat      cout
newidcirc
39001#                   0.069358
39002#                   0.090776
39003#                   0.080744
frontière                0.102645
```

On peut regarder la distribution de ce coût sur l'ensemble des circonscriptions.

[107]:
```
aggall = agg_circonscription(t1t2noz)
aggall["ratio"] = aggall["SOC"] / aggall["total"]
aggall["cout"] = aggall["ratio"].apply(fonction_cout)
```

[108]:
```
fig, axes = plt.subplots(1, 2, figsize=(12,4))
aggall.hist("ratio", ax=axes[0], bins=50)
aggall.hist("cout", ax=axes[1], bins=50);
```



Il faut donc minimiser la somme des coûts pour chaque circonscription. Comment utiliser ce coût au niveau de chaque bureau ?

1.7.5 Propager la fonction de coût au niveau de chaque bureau

Le coût se calcule au niveau de chaque circonscription. On en considère deux : C_1 et C_2 et un bureau $b \in C_1$. On calcule le terme :

$$\Delta(b, C_1, C_2) = C(C_1 \backslash \{b\}) + C(C_2 \cup \{b\}) - (C(C_1) + C(C_2))$$

Il correspond à la différence des coûts pour les deux circonscriptions obtenus en changeant le bureau b de circonscription. Si $\Delta(b, C_1, C_2) < 0$, le coût de départ est plus élevé que le coût après modifications. C'est ce qu'on cherche.

[109]:
```
def compute_cost(thewinner, bex, asso):
    """
    calcule les detta pour chaque bureau dans subset

    Parameters
    ----------

    thewinner: le parti à favoriser

    bex: liste de dictionnaires
         ``[{'SOC': 146, 'UMP': 175, 'idbureau3': '3900133***#', 'nbwinner': 175,␣
 ↪'total': 321, 'winner': 'UMP'}]``

    asso: dictionnaire { bureau: circonscription }

    Returns
    -------

    dictionnaire ``{idcirc : coût}``
    """
    count = {}
    total = {}
    for d in bex:
        circ = asso[d["idbureau3"]]
        if circ not in count:
```

```
            count[circ] = 0
            total[circ] = 0
        count[circ] += d.get(thewinner, 0)
        total[circ] += d["total"]
    cout = {k: fonction_cout(count[k] / total[k]) for k in count}
    return cout

compute_cost("SOC", bex, asso)
```

[109]:
```
{'39003#': 0.08173278657257299,
 '39001#': 0.07143124298715875,
 '39002#': 0.09608555286521392}
```

[110]:
```
def compute_delta(thewinner, bex, asso, subset):
    """
    calcule les detta pour chaque bureau dans subset

    Parameters
    ----------

    thewinner: le parti à favoriser

    bex: liste de dictionnaires
         ``[{'SOC': 146, 'UMP': 175, 'idbureau3': '3900133***#', 'nbwinner': 175,␣
 ↪'total': 321, 'winner': 'UMP'}]``

    asso: dictionnaire { bureau: circonscription }

    subset: dictionnaire de tuple ``(idbureau3, idbureau3) : (distance, circ1, circ2)``

    Returns
    -------

    delta pour chaque couple dans subset ``(idbureau3, idbureau3) : (distance, circ1,␣
 ↪circ2, delta)``
    """
    cout0 = compute_cost(thewinner, bex, asso)
    res = {}
    for k, v in subset.items():
        idb1, idb2 = k
        dist, c1, c2 = v
        if asso[idb1] != c1:
            raise Exception("inattendu: c1={0} c2={1}\n{2} : {3}".format(asso[idb1],␣
 ↪asso[idb2],k,v))
        if asso[idb2] != c2:
            raise Exception("inattendu")
        asso[idb1], asso[idb2] = asso[idb2], asso[idb1]
        cout1 = compute_cost(thewinner, bex, asso)
        asso[idb1], asso[idb2] = asso[idb2], asso[idb1]
        delta = sum(cout1.values()) - sum(cout0.values())
        res[k] = (dist, c1, c2, delta)
    return res
```

```
zero_dist_39 = {k:v for k,v in dist_39.items() if v[0] == 0}
res = compute_delta('SOC', bex, asso, zero_dist_39)
list(res.items())[:2]
```

[110]:
```
[(('3900201***#', '3900323***#'),
  (0.0, '39003#', '39001#', -6.056414690147616e-06)),
 (('3900201***#', '3902823***#'),
  (0.0, '39003#', '39001#', -2.3907330787165115e-05))]
```

[111]:
```
df = pandas.DataFrame([dict(idb1=k[0], idb2=k[1], c1=v[1], c2=v[2], delta=v[3]) for
↪k,v in res.items()])
df = df.sort_values("delta")
df.head()
```

[111]:

	idb1	idb2	c1	c2	delta
44	3911601***#	3954006***#	39003#	39002#	-0.000356
141	3954006***#	3911601***#	39002#	39003#	-0.000356
76	3930716***#	3939721***#	39002#	39001#	-0.000329
108	3939721***#	3930716***#	39001#	39002#	-0.000329
124	3944601***#	3943423***#	39003#	39001#	-0.000287

[112]:
```
negative = df[df["delta"] < 0]
negative.head()
```

[112]:

	idb1	idb2	c1	c2	delta
44	3911601***#	3954006***#	39003#	39002#	-0.000356
141	3954006***#	3911601***#	39002#	39003#	-0.000356
76	3930716***#	3939721***#	39002#	39001#	-0.000329
108	3939721***#	3930716***#	39001#	39002#	-0.000329
124	3944601***#	3943423***#	39003#	39001#	-0.000287

[113]:
```
def asso2solution(asso):
    solution = {}
    for k, v in asso.items():
        if v not in solution:
            solution[v] = [k]
        else:
            solution[v].append(k)
    return solution
set(asso2solution(asso))
```

[113]:
```
{'39001#', '39002#', '39003#'}
```

[114]:
```
agg = agg_circonscription(res39, solution=asso2solution(asso), col_circ=None,
↪col_place="idbureau3")
agg["ratio"] = agg["SOC"] / agg["total"]
agg["cout"] = agg["ratio"].apply(fonction_cout)
agg
```

[114]:

Code nuance du candidat	SOC	UMP	winner	nbwinner	total	ratio	\
new_circ_temp							
39001#	19193	20912	UMP	20912	40105	0.478569	
39002#	14060	16915	UMP	16915	30975	0.453914	

```
39003#                   19641   22303    UMP      22303   41944   0.468267

Code nuance du candidat      cout
new_circ_temp
39001#                   0.071431
39002#                   0.096086
39003#                   0.081733
```

On change un bureau.

[115]:
```
asso2 = asso.copy()
asso2["3954006***#"], asso2["3911601***#"] = asso2["3911601***#"], asso2["3954006***#"]
agg = agg_circonscription(res39, solution=asso2solution(asso2), col_circ=None,␣
 ↪col_place="idbureau3")
agg["ratio"] = agg["SOC"] / agg["total"]
agg["cout"] = agg["ratio"].apply(fonction_cout)
agg
```

[115]:
```
Code nuance du candidat    SOC    UMP winner  nbwinner  total     ratio  \
new_circ_temp
39001#                   19193   20912    UMP      20912   40105   0.478569
39002#                   14098   16867    UMP      16867   30965   0.455288
39003#                   19603   22351    UMP      22351   41954   0.467250

Code nuance du candidat      cout
new_circ_temp
39001#                   0.071431
39002#                   0.094712
39003#                   0.082750
```

On en change un peu plus.

[116]:
```
pairs = [tuple(sorted([_["idb1"], _["idb2"]])) for _ in negative[["idb1", "idb2"]].
 ↪to_dict("records")]
len(pairs)
```

[116]:
```
72
```

[117]:
```
done = {} # on s'assure qu'on déplace un bureau qu'une seule fois
          # à la frontière entre 3 départements
asso2 = asso.copy()
for k1, k2 in pairs:
    if k1 in done or k2 in done:
        continue
    asso2[k1], asso2[k2] = asso2[k2], asso2[k1]
    done[k1] = k2
    done[k2] = k1

    agg = agg_circonscription(res39, solution=asso2solution(asso2), col_circ=None,␣
 ↪col_place="idbureau3")
    agg["ratio"] = agg["SOC"] / agg["total"]
    agg["cout"] = agg["ratio"].apply(fonction_cout)
    print(len(done), agg["cout"].sum())
```

```
2 0.2488931692735059
4 0.248586899182697
6 0.24826066162932847
8 0.24814890661798567
10 0.24804819971253794
12 0.24796509713613613
14 0.24790247728049586
16 0.24784200900478276
18 0.2478015215921256
20 0.24774025599324034
22 0.24771405047988282
24 0.24770659236526615
26 0.24772330324213526
```

[118]:
```
agg = agg_circonscription(res39, solution=asso2solution(asso2), col_circ=None,␣
 ↪col_place="idbureau3")
agg["ratio"] = agg["SOC"] / agg["total"]
agg["cout"] = agg["ratio"].apply(fonction_cout)
agg
```

[118]:
```
Code nuance du candidat    SOC    UMP winner  nbwinner  total     ratio  \
new_circ_temp
39001#                   17990  19851    UMP     19851  37841  0.475410
39002#                   14365  16965    UMP     16965  31330  0.458506
39003#                   20539  23314    UMP     23314  43853  0.468360

Code nuance du candidat      cout
new_circ_temp
39001#                   0.074590
39002#                   0.091494
39003#                   0.081640
```

On vérifie sur une carte.

[119]:
```
subset = [k for k, v in dist_39.items() if v[0] == 0]
subset = set([_[0] for _ in subset] + [_[1] for _ in subset])
map_osm = carte_interactive(bex, asso2, new_shape_bureau, flag_bureau=subset,␣
 ↪choix=choix)
folium_html_map(map_osm, width="70%")
```

```
{'idbureau3': '3900133***#', 'SOC': 146, 'UMP': 175, 'winner': 'UMP',
'nbwinner': 175, 'total': 321}
```

```
c:\python372_x64\lib\site-packages\folium\folium.py:426: FutureWarning: The
choropleth  method has been deprecated. Instead use the new Choropleth class,
which has the same arguments. See the example notebook 'GeoJSON_and_choropleth'
for how to do this.
  FutureWarning
```

[119]:
```
<pyensae.notebookhelper.folium_helper.folium_html_map.<locals>.CustomFoliumMap
at 0x1cb5b553748>
```

On s'aperçoit que les frontières ne sont plus très linéaires. Toutefois, en répétant ce processus plusieurs fois, nous devrions être capables de faire évoluer les scores. Une autre option consiste à reconstruire les circonscription en considérant le département comme une page blanche via un mécanisme plus proche d'une classification ascendante hiérarchique.

1.8 Solution globale

Cette solution reprend les éléments présentés dans ce notebook mais sera implémentée dans un programme séparé. Cet exercice atteint les limites de ce qu'un notebook peut contenir.

[120]: