seance6_graphes_enonce

July 2, 2023

1 Graphes - énoncé

Ce notebook introduit matplotlib et d'autres modules Python qui permettent de tracer des graphes et bâtis sur la même logique que matplotlib.
Pour avoir des graphiques inclus dans le notebook, il faut ajouter cette ligne et l'exécuter en premier.

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

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

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

[3]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[3]:
```
<IPython.core.display.HTML object>
```

1.1 Données

Pour tous les exemples qui suivent, on utilise les résultat élection présidentielle de 2012. Si vous n'avez pas le module actuariat_python, il vous suffit de recopier le code de la fonction elections_presidentielles qui utilise la fonction read_excel. La fonction utilise des données stockées localement afin que le code ci-dessous fonctionne toujours même si le format des données change sur le site *data.gouv.fr*.

[4]:
```
from actuariat_python.data import elections_presidentielles
dict_df = elections_presidentielles(local=True, agg="dep")
```

[5]:
```
list(dict_df.keys())
```

[5]:
```
['circ1', 'circ2', 'dep1', 'dep2']
```

[6]:
```
dict_df["dep1"].head()
```

[6]:
```
   Code du département   Libellé du département  Code de la circonscription  \
0                    1                      AIN                          15
1                    2                    AISNE                          15
2                    3                   ALLIER                           6
3                    4  ALPES-DE-HAUTE-PROVENCE                           3
4                    5             HAUTES-ALPES                           3

   Inscrits  Votants  Exprimés  Blancs et nuls  Nathalie ARTHAUD (LO)  \
```

```
0    393808   327812    321359            6453                    1794
1    376068   303140    297944            5196                    2490
2    256275   211009    205950            5059                    1482
3    123933   102899    100788            2111                     487
4    106865    88619     86777            1842                     488

   Philippe POUTOU (NPA)  Jean-Luc MELENCHON (FG)  François HOLLANDE (PS)  \
0                   3323                    30898                   73096
1                   3860                    30360                   80751
2                   2584                    27969                   61131
3                   1394                    15269                   24551
4                   1152                    12175                   21248

   Eva JOLY (EELV)  François BAYROU (MODEM)  Nicolas SARKOZY (UMP)  \
0             7268                    32650                  97722
1             3455                    19895                  72090
2             3232                    17814                  49477
3             2933                     7483                  25668
4             3147                     8559                  22655

   Nicolas DUPONT-AIGNAN (DLR)  Marine LE PEN (FN)  Jacques CHEMINADE (SP)
0                         7208               66540                     860
1                         5853               78452                     738
2                         4068               37736                     457
3                         1845               20875                     283
4                         1782               15359                     212
```

[7]:
```python
dict_df["dep2"].head()
```

[7]:
```
  Code du département            Libellé du département  \
0                 979  SAINT-BARTHELEMY et SAINT-MARTIN
1                  01                               AIN
2                  02                             AISNE
3                  03                            ALLIER
4                  04           ALPES-DE-HAUTE-PROVENCE

   Code de la circonscription  Inscrits  Votants  Exprimés  Blancs et nuls  \
0                           1     22686     9907      9492             415
1                          15    393866   326587    307074           19513
2                          15    376073   302076    281020           21056
3                           6    256211   211132    196208           14924
4                           3    123895   103581     96942            6639

   François HOLLANDE (PS)  Nicolas SARKOZY (UMP)
0                    3851                   5641
1                  131333                 175741
2                  147260                 133760
3                  111615                  84593
4                   49498                  47444
```

On corrige le code du département 01 –> 1.

[8]:
```python
def cleandep(s):
    if isinstance(s, str):
```

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

[9]:
```
deps = dict_df["dep1"].merge(dict_df["dep2"],
                                        on="Code du département",
                                        suffixes=("T1", "T2"))
deps.columns
```

[9]:
```
Index(['Code du département', 'Libellé du départementT1',
       'Code de la circonscriptionT1', 'InscritsT1', 'VotantsT1', 'ExprimésT1',
       'Blancs et nulsT1', 'Nathalie ARTHAUD (LO)', 'Philippe POUTOU (NPA)',
       'Jean-Luc MELENCHON (FG)', 'François HOLLANDE (PS)T1',
       'Eva JOLY (EELV)', 'François BAYROU (MODEM)', 'Nicolas SARKOZY (UMP)T1',
       'Nicolas DUPONT-AIGNAN (DLR)', 'Marine LE PEN (FN)',
       'Jacques CHEMINADE (SP)', 'Libellé du départementT2',
       'Code de la circonscriptionT2', 'InscritsT2', 'VotantsT2', 'ExprimésT2',
       'Blancs et nulsT2', 'François HOLLANDE (PS)T2',
       'Nicolas SARKOZY (UMP)T2'],
      dtype='object')
```

[10]:
```
deps["rHollandeT1"] = deps['François HOLLANDE (PS)T1'] / (deps["VotantsT1"] -␣
 ↪deps["Blancs et nulsT1"])
deps["rSarkozyT1"] = deps['Nicolas SARKOZY (UMP)T1'] / (deps["VotantsT1"] -␣
 ↪deps["Blancs et nulsT1"])
deps["rNulT1"] = deps["Blancs et nulsT1"] / deps["VotantsT1"]
deps["rHollandeT2"] = deps["François HOLLANDE (PS)T2"] / (deps["VotantsT2"] -␣
 ↪deps["Blancs et nulsT2"])
deps["rSarkozyT2"] = deps['Nicolas SARKOZY (UMP)T2'] / (deps["VotantsT2"] -␣
 ↪deps["Blancs et nulsT2"])
deps["rNulT2"] = deps["Blancs et nulsT2"] / deps["VotantsT2"]
data = deps[["Code du département", "Libellé du départementT1",
             "VotantsT1", "rHollandeT1", "rSarkozyT1", "rNulT1",
             "VotantsT2", "rHollandeT2", "rSarkozyT2", "rNulT2"]]
data_elections = data # parfois data est remplacé dans la suite
data.head()
```

[10]:
```
  Code du département Libellé du départementT1  VotantsT1  rHollandeT1  \
0                   1                      AIN     327812     0.227459
1                   2                    AISNE     303140     0.271027
2                   3                   ALLIER     211009     0.296824
3                   4  ALPES-DE-HAUTE-PROVENCE     102899     0.243591
4                   5             HAUTES-ALPES      88619     0.244858

   rSarkozyT1    rNulT1  VotantsT2  rHollandeT2  rSarkozyT2    rNulT2
0    0.304090  0.019685     326587     0.427692    0.572308  0.059748
1    0.241958  0.017141     302076     0.524020    0.475980  0.069704
2    0.240238  0.023975     211132     0.568861    0.431139  0.070686
```

```
3    0.254673  0.020515     103581     0.510594    0.489406  0.064095
4    0.261071  0.020786      89405     0.508935    0.491065  0.067390
```

[11]:
```
deps.to_excel("deps.xlsx")
dict_df["dep1"].to_excel("T1.xlsx")
dict_df["dep2"].to_excel("T2.xlsx")
```

1.2 De pandas à matplotlib

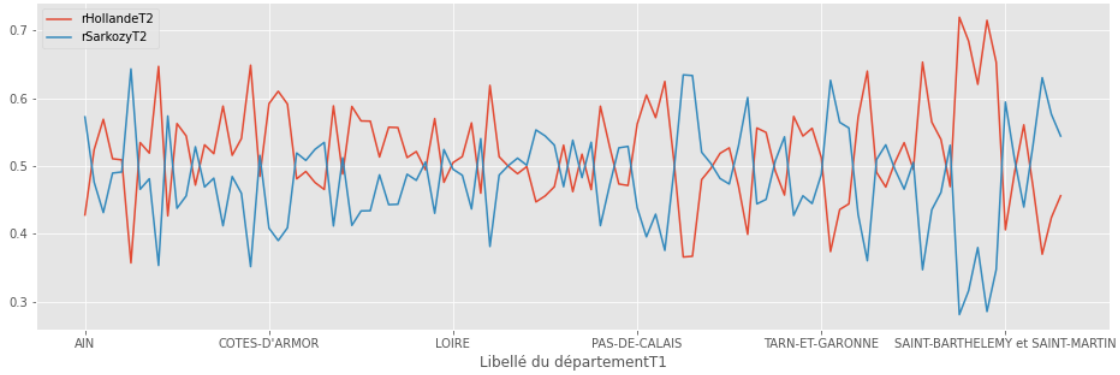
Lorsqu'on construit un graphique avec des données stockées dans un DataFrame, on suit généralement le processus suivant :

- Voir si un graphique correspond dans la page visualisation de pandas
- Voir la gallerie de matplotlib
- Chercher un exemple de graphique sur un moteur de recherche pour tomber sur une page comme celle-ci Using Python libraries to plot two horizontal bar charts sharing same y axis
- Assembler différentes sources

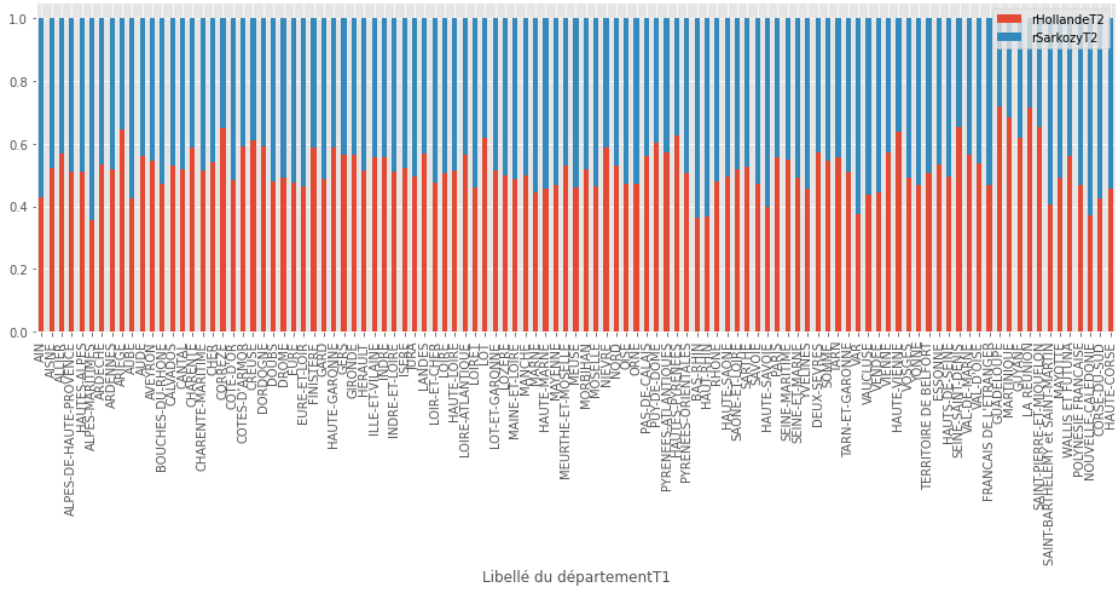
1.2.1 plot

La méthode plot permet de faire la plupart des graphiques standards (voir Plotting).

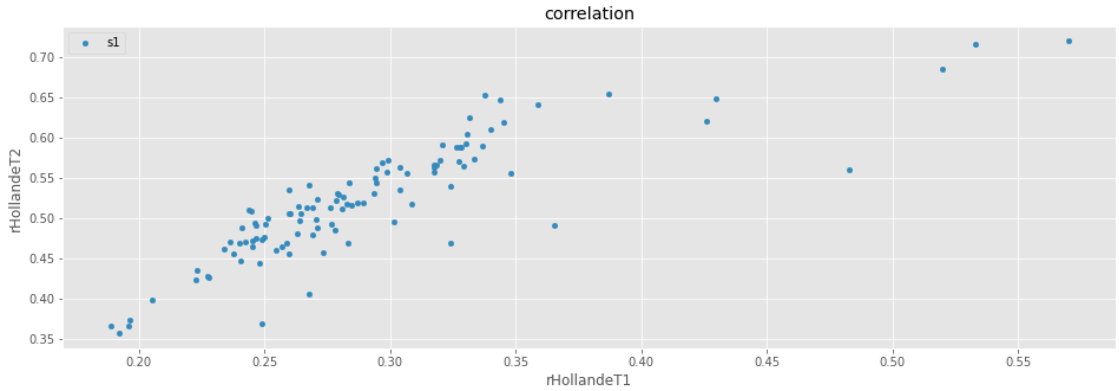
[12]:
```
data.plot(x="Libellé du départementT1", y=["rHollandeT2", "rSarkozyT2"],␣
↪figsize=(16,5));
```



[13]:
```
data.plot(x="Libellé du départementT1", y=["rHollandeT2", "rSarkozyT2"],
          figsize=(16,5), kind="bar", stacked=True);
```



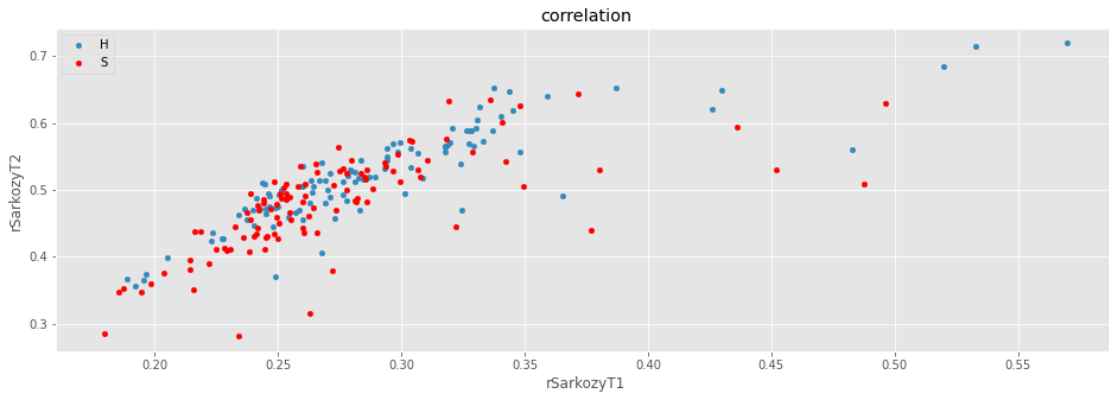
[14]:
```
data.plot(x="rHollandeT1", y="rHollandeT2", figsize=(16,5),
          kind="scatter", label="s1", title="correlation");
```



1.2.2 superposition

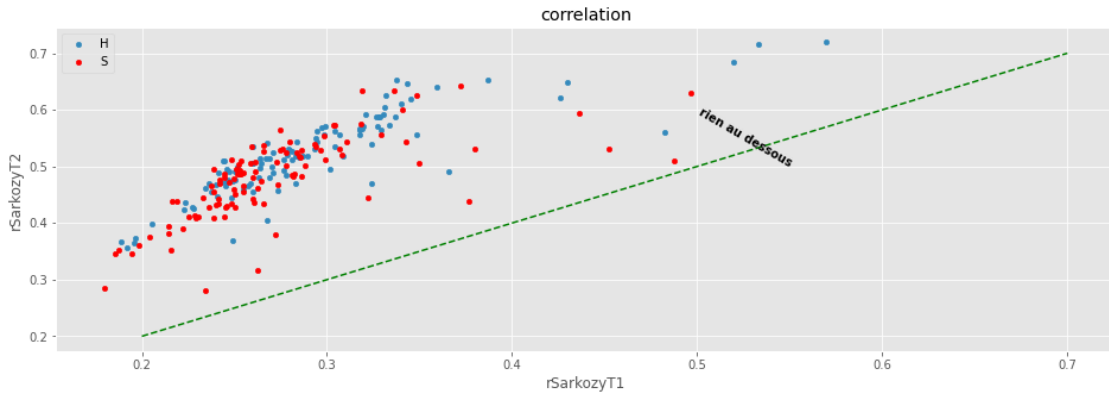
La méthode plot retourne un objet de type Axes. On peut superposer plusieurs courbes sur le même graphique en s'assurant que la seconde courbe utilise le même objet.

[15]:
```
ax = data.plot(x="rHollandeT1", y="rHollandeT2", figsize=(16,5),
               kind="scatter", label="H", title="correlation")
data.plot(x="rSarkozyT1", y="rSarkozyT2", kind="scatter", label="S", ax=ax, c="red");
```



On ajoute une ligne avec la méthode Axes.plot ou du text avec text :

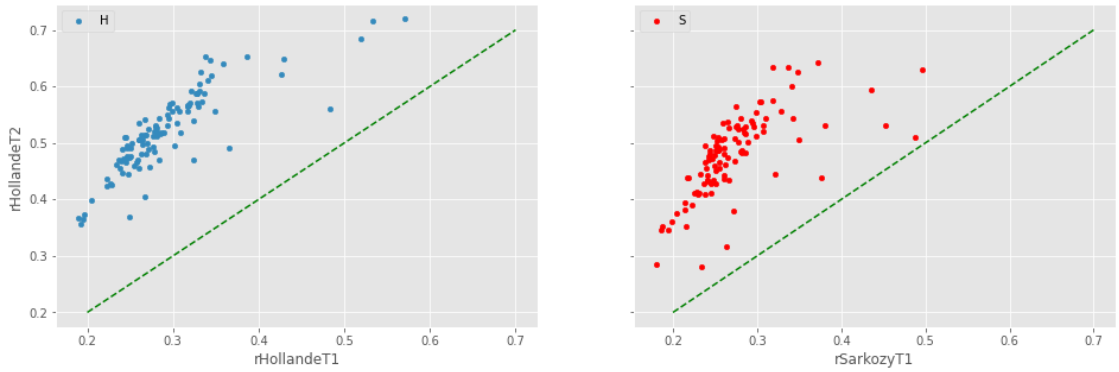
[16]:
```
ax = data.plot(x="rHollandeT1", y="rHollandeT2", figsize=(16,5),
               kind="scatter", label="H", title="correlation")
data.plot(x="rSarkozyT1", y="rSarkozyT2", kind="scatter", label="S", ax=ax, c="red")
ax.plot([0.2,0.7], [0.2,0.7], "g--")
ax.text(0.5, 0.5, "rien au dessous", weight="bold", rotation="-30");
```



1.2.3 plusieurs graphes sur la même figure

pandas crée une Figure de façon implicite avec un seul graphe. Pour créer plusieurs graphes, il faut créer ce type d'objet en précisant qu'il y aura plusieurs Axes avec la fonction subplots et les transmettre à *pandas*. On peut également partager l'axe des X ou l'axe des Y.

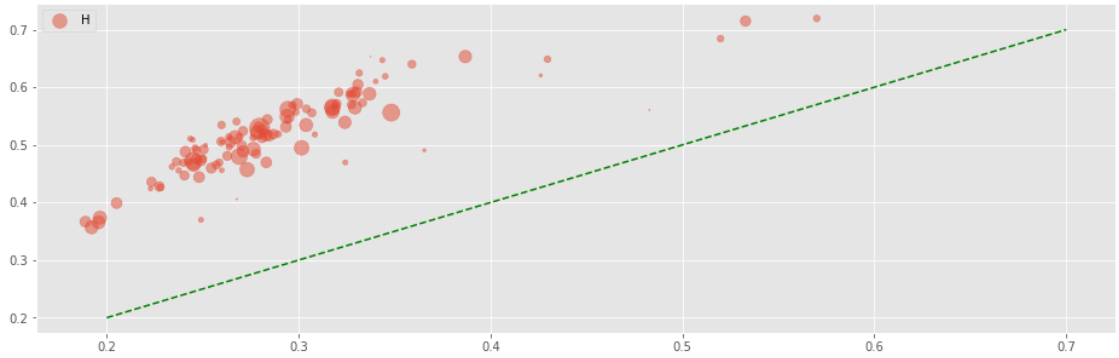
[17]:
```
import matplotlib.pyplot as plt
fig, axes = plt.subplots(1, 2, figsize=(16,5), sharey=True)
data.plot(x="rHollandeT1", y="rHollandeT2", figsize=(16,5),
          kind="scatter", label="H", ax=axes[0])
data.plot(x="rSarkozyT1", y="rSarkozyT2", kind="scatter",
          label="S", ax=axes[1], c="red")
axes[0].plot([0.2,0.7], [0.2,0.7], "g--")
axes[1].plot([0.2,0.7], [0.2,0.7], "g--");
```



1.2.4 matplotlib sans pandas

On peut se passer de pandas et s'inspirer d'un graphe de la gallerie pour ajouter des points dépendants du nombre de votants scatter_demo et ajouter une légende manuellement avec la méthode legend.

[18]:
```
import matplotlib.pyplot as plt
fig, axes = plt.subplots(1, 1, figsize=(16,5))
c = axes.scatter(x=data["rHollandeT1"],
            y=data["rHollandeT2"],
            s=data["VotantsT1"]/5000, alpha=0.5)
axes.plot([0.2,0.7], [0.2,0.7], "g--")
axes.legend( (c,), ("H",) );
```

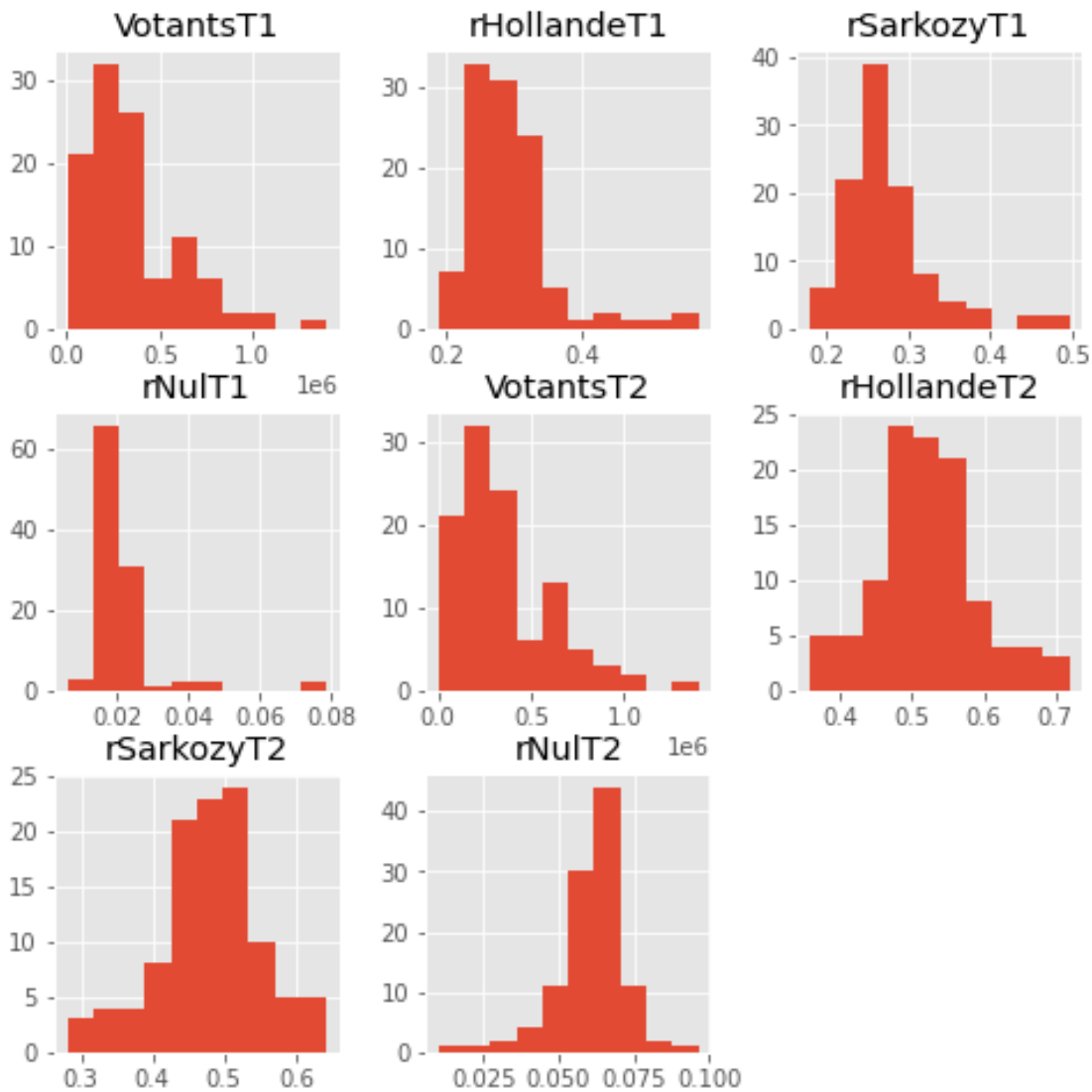


1.3 Pandas et graphes prêts à l'emploi

1.3.1 histogrammes

avec hist

[19]:
```
data.hist(figsize=(8, 8));
```



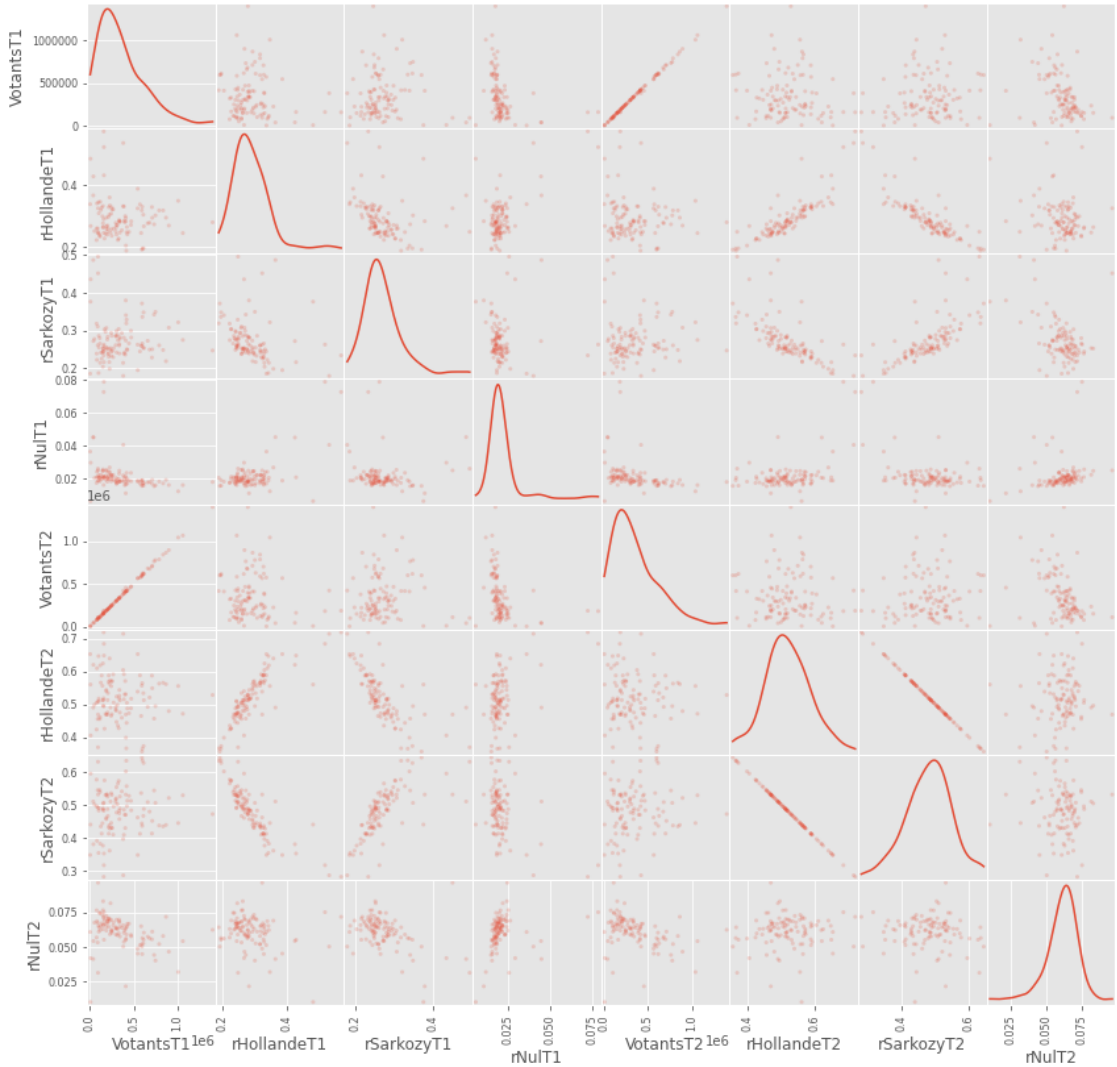
Le paramètre *figsize* permettrait de modifier la taille du graphique.

1.3.2 correlations

avec scatter_matrix

[20]:
```
from pandas.plotting import scatter_matrix
scatter_matrix(data, alpha=0.2, figsize=(14, 14), diagonal='kde')
print("-");
```

-



1.4 cartes avec cartopy

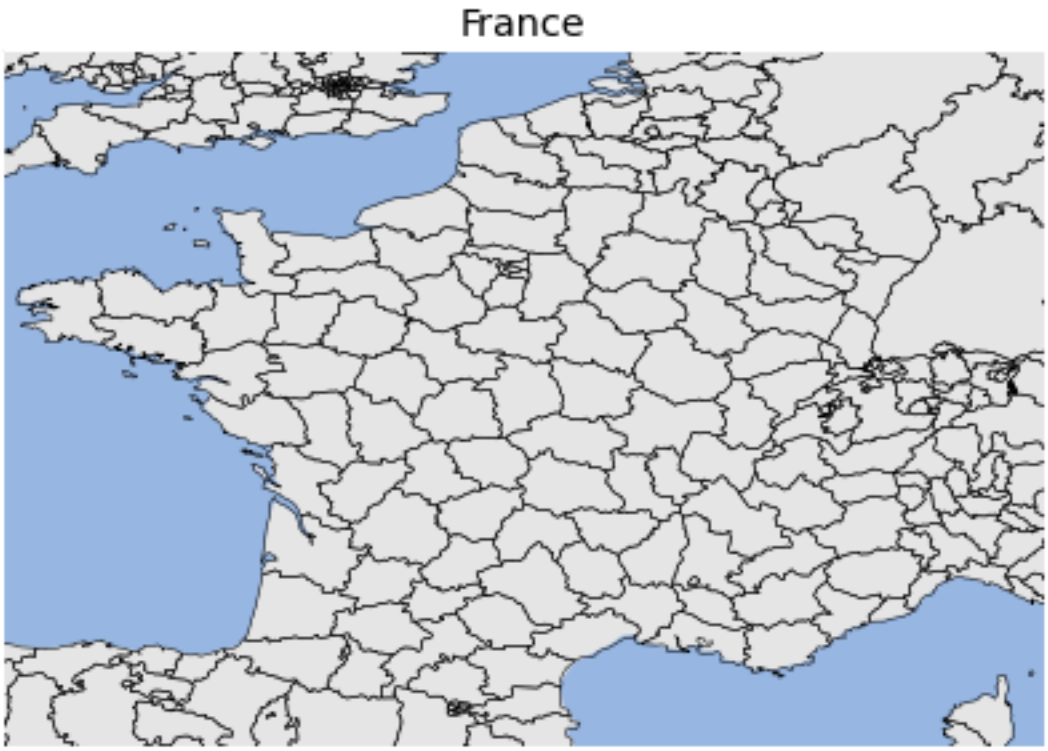
Les coordonnées sur une carte se font avec des coordonnées géographiques : longitude et latitude. La distance entre deux lieux géographiques se calcule grâce à la distance de Haversine. Les graphes se font avec cartopy.

1.4.1 une carte simple

On la choisit centrée sur la France. La carte se dessine avec matplotlib auquel cartopy ajoute un système de projection différent. Comme elle accepte un argument `ax`, il est possible de changer sa taille ou de la juxtaposer à côté d'un autre graphe. Le module cartopy ne contient pas toutes les informations sur le territoire français et certaines options ne semble pas avoir d'effet.

[21]:
```
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(7,7))
```

```
ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree())
ax.set_extent([-5, 10, 42, 52])

ax.add_feature(cfeature.OCEAN.with_scale('50m'))
# ax.add_feature(cfeature.COASTLINE)
# ax.add_feature(cfeature.RIVERS)  # pas d'effet, cartopy ne connaît pas les rivières␣
↪en France
# ax.add_feature(cfeature.BORDERS, linestyle=':')
# cette instruction télécharge un fichier (10m=14Mo, 50m, 110m)
# il faut la résolution 10m pour la France
ax.add_feature(cfeature.STATES.with_scale('10m'))
ax.set_title('France');
```



La méthode with_scale propose trois résolution 10m, 50m, 110m. La module cartopy n'inclut que la résolution 110m, le reste doit être téléchargé.

1.4.2 exercice 1 : centrer la carte de la France

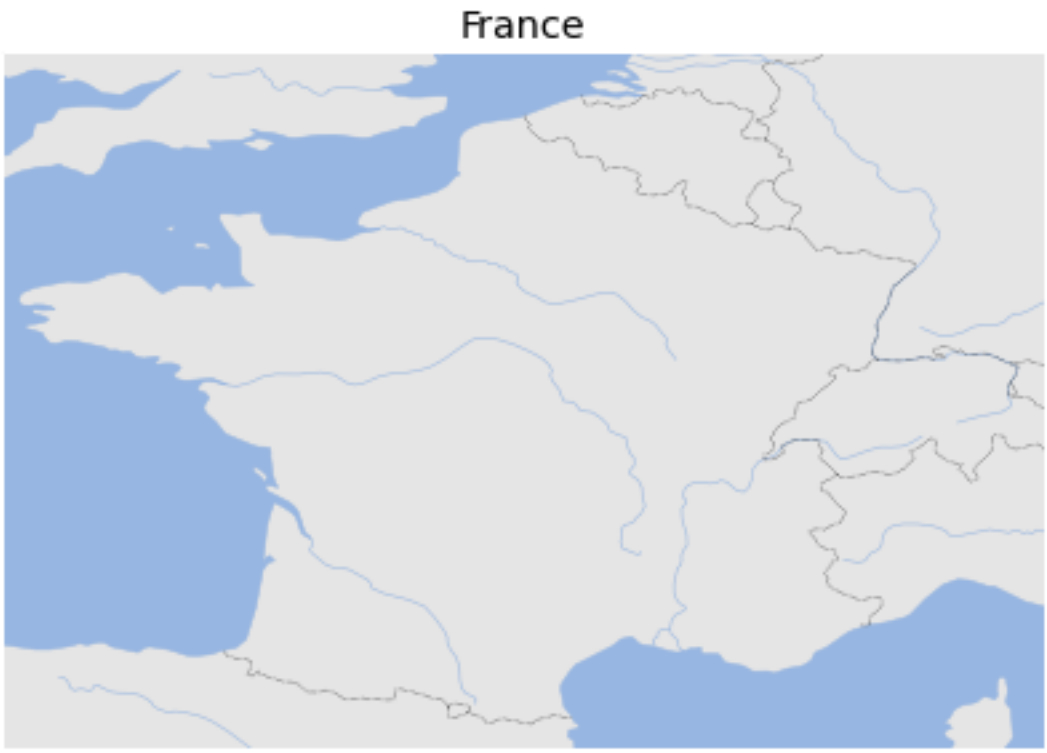
[22] :

```
```

1.4.3 ajouter du texte ou une marque

Sur une carte, on veut la plupart du temps ajouter du texte. On reprend le début de ce code qu'on place dans une fonction, puis on place Paris. On utilise pour cela les fonctions standard de *matplotlib* mais on convertit les coordonnées géographiques en coordonnées relatives au graphe (donc dans un repère différent).

[23]:
```
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.pyplot as plt

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, 42, 52])

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

carte_france();
```



[24]:
```
import matplotlib.pyplot as plt

ax = carte_france()

lon = 2.3488000
lat = 48.853410
ax.plot([lon], [lat], 'ro', markersize=6)
ax.text(lon, lat, "Paris");
```

On connaît rarement les coordonnées de chaque ville mais un moteur de recherche donne rapidement des pistes pour trouver ces données. Il faut néanmoins s'assurer que le licence autorise ce qu'on l'intention de faire avec :
Liste des villes de France en SQL, CSV ou XML

[25]:
```
from pyensae.datasource import download_data
download_data("villes_france.csv", url="http://sql.sh/ressources/sql-villes-france/")
```

[25]: 'villes_france.csv'

[26]:
```
cols = ["ncommune", "numero_dep", "slug", "nom", "nom_simple", "nom_reel",
        "nom_soundex", "nom_metaphone", "code_postal",
        "numero_commune", "code_commune", "arrondissement", "canton",
        "pop2010", "pop1999", "pop2012",
        "densite2010", "surface", "superficie", "dlong", "dlat", "glong",
        "glat", "slong", "slat", "alt_min", "alt_max"]
import pandas
df = pandas.read_csv("villes_france.csv", header=None,low_memory=False, names=cols)
```

[27]:
```
df.head()
```

[27]:
```
   ncommune numero_dep                   slug                     nom  \
0         1         01                   ozan                    OZAN
1         2         01  cormoranche-sur-saone  CORMORANCHE-SUR-SAONE
2         3         01              plagne-01                  PLAGNE
3         4         01                tossiat                 TOSSIAT
4         5         01               pouillat                POUILLAT
```

```
              nom_simple                nom_reel nom_soundex nom_metaphone  \
0                   ozan                    Ozan        O250           OSN
1  cormoranche sur saone  Cormoranche-sur-Saône   C65652625    KRMRNXSRSN
2                 plagne                  Plagne        P425          PLKN
3                tossiat                 Tossiat        T230           TST
4               pouillat                Pouillat        P430           PLT

  code_postal  numero_commune  … surface  superficie    dlong     dlat  \
0       01190             284  …      93        6.60  4.91667  46.3833
1       01290             123  …     107        9.85  4.83333  46.2333
2       01130             298  …      20        6.20  5.73333  46.1833
3       01250             422  …     138       10.17  5.31667  46.1333
4       01250             309  …      14        6.23  5.43333  46.3333

    glong     glat    slong      slat  alt_min  alt_max
0  2866.0  51546.0  45456.0  462330.0    170.0    205.0
1  2772.0  51379.0  44953.0  461427.0    168.0    211.0
2  3769.0  51324.0  54342.0  461131.0    560.0    922.0
3  3309.0  51268.0  51854.0  460828.0    244.0    501.0
4  3435.0  51475.0  52542.0  461938.0    333.0    770.0

[5 rows x 27 columns]
```

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

[28] :

```
```

1.4.5 départements

Pour dessiner des formes sur une carte, il faut connaître les coordonnées de ces formes. L'article suivant Matplotlib Basemap tutorial 10: Shapefiles Unleached, continued permet de dessiner les départements belges. On va s'en inspirer pour dessiner les départements français. La première chose à faire est de récupérer des données géographiques. Une façon simple de les trouver est d'utiliser un moteur de recherche avec le mot clé **shapefile** inclus dedans : c'est le format du fichier. *shapefile france* permet d'obtenir quelques sources. En voici d'autres :

- GADM : database of Global Administrative Areas
- OpenData.gouv commune : base de données sur data.gouv.fr
- The National Map Small-Scale Collection : Etats-Unis
- ArcGIS : API Javascripts
- Natural Earth : Natural Earth is a public domain map dataset available at 1:10m, 1:50m, and 1:110 million scales. Featuring tightly integrated vector and raster data, with Natural Earth you can make a variety of visually pleasing, well-crafted maps with cartography or GIS software.
- thematicmapping : World Borders Dataset
- OpenStreetMap Data Extracts : OpenStreetMap data
- OpenStreetMapData : OpenStreetMap data
- Shapefile sur Wikipedia : contient divers liens vers des sources de données

La première chose à vérifier est la licence associées aux données : on ne peut pas en faire ce qu'on veut. Pour cet exemple, j'ai choisi la première source de données, GADM. La licence n'est pas précisée explicitement (on peut trouver *happy to share* sur le site, la page wikipedia GADM précise : *GADM is not freely available for commercial use. The GADM project created the spatial data for many countries from spatial databases*

provided by national governments, NGO, and/or from maps and lists of names available on the Internet (e.g. from Wikipedia). C'est le choix que j'avais fait en 2015 mais l'accès à ces bases a probablement changé car l'accès est restreint. J'ai donc opté pour les bases accessibles depuis data.gouv.fr. Leur seul inconvénient est que les coordonnées sont exprimées dans une projection de type Lambert 93. Cela nécessite une conversion.

[29]:
```
from pyensae.datasource import download_data
try:
    download_data("GEOFLA_2-1_DEPARTEMENT_SHP_LAMB93_FXX_2015-12-01.7z",
                  website="https://wxs-telechargement.ign.fr/oikr5jryiph0iwhw36053ptm/
 ↪telechargement/inspire/" + \
                ␣
 ↪"GEOFLA_THEME-DEPARTEMENTS_2015_2$GEOFLA_2-1_DEPARTEMENT_SHP_LAMB93_FXX_2015-12-01/
 ↪file/")
except Exception as e:
    # au cas le site n'est pas accessible
    download_data("GEOFLA_2-1_DEPARTEMENT_SHP_LAMB93_FXX_2015-12-01.7z", website="xd")
```

[30]:
```
from pyquickhelper.filehelper import un7zip_files
try:
    un7zip_files("GEOFLA_2-1_DEPARTEMENT_SHP_LAMB93_FXX_2015-12-01.7z",␣
 ↪where_to="shapefiles")
    departements = 'shapefiles/GEOFLA_2-1_DEPARTEMENT_SHP_LAMB93_FXX_2015-12-01/GEOFLA/
 ↪1_DONNEES_LIVRAISON_2015/' + \
                    'GEOFLA_2-1_SHP_LAMB93_FR-ED152/DEPARTEMENT/DEPARTEMENT.shp'
except FileNotFoundError as e:
    # Il est possible que cette instruction ne fonctionne pas.
    # Dans ce cas, on prendra une copie de ce fichier.
    import warnings
    warnings.warn("Plan B parce que " + str(e))
    download_data("DEPARTEMENT.zip")
    departements = "DEPARTEMENT.shp"

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

La license accompagne les données : *ce produit est téléchargeable et utilisable gratuitement sous licence Etalab.* Pour un usage commercial, il faut faire attentation à la license associée aux données. Le seul inconvénient des données *GEOFLA* est que certaines sont données dans le système de coordonnées Lambert 93 (voir aussi Cartographie avec R).

[31]:
```
shp = departements
import shapefile
r = shapefile.Reader(shp)
shapes = r.shapes()
records = r.records()
len(shapes), len(records)
```

[31]:
```
(96, 96)
```

[32]:
```
r.bbox
```

[32]:
```
[99217.1, 6049646.300000001, 1242417.2, 7110480.100000001]
```

On regarde une zone en particulier mais on réduit la quantité de données affichées :

[33]:
```
d = shapes[0].__dict__.copy()
d["points"] = d["points"][:10]
d
```

[33]:
```
{'shapeType': 5,
 'points': [(701742.0, 6751181.100000001),
  (701651.9, 6751166.9),
  (701552.0, 6751162.7),
  (700833.7000000001, 6751313.7),
  (700669.4, 6751380.0),
  (700475.4, 6751476.600000001),
  (700400.7000000001, 6751517.2),
  (700098.3, 6751789.600000001),
  (699993.8, 6751845.4),
  (699874.1000000001, 6751876.4)],
 'parts': [0],
 'bbox': [688654.4, 6690595.300000001, 800332.3, 6811114.5]}
```

350 départements, sûr ?

[34]:
```
records[0], records[1]
```

[34]:
```
(Record #0: ['DEPARTEM0000000000000004', '89', 'YONNE', '024', 'AUXERRE',
742447, 6744261, 748211, 6750855, '27', 'BOURGOGNE-FRANCHE-COMTE'],
 Record #1: ['DEPARTEM0000000000000028', '69', 'RHONE', '381', 'LYON', 842221,
6520526, 832095, 6530600, '84', 'AUVERGNE-RHONE-ALPES'])
```

[35]:
```
len(set([r[6] for r in records]))
```

[35]:
```
96
```

Puis je récupère le code final (toujours à Matplotlib Basemap tutorial 10: Shapefiles Unleached, continued) en l'adaptant pour la France. Petite astuce, on utilie la fonction lambert932WGPS du module ensae_teaching_cs. On recopie le code ici :

[36]:
```
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
```

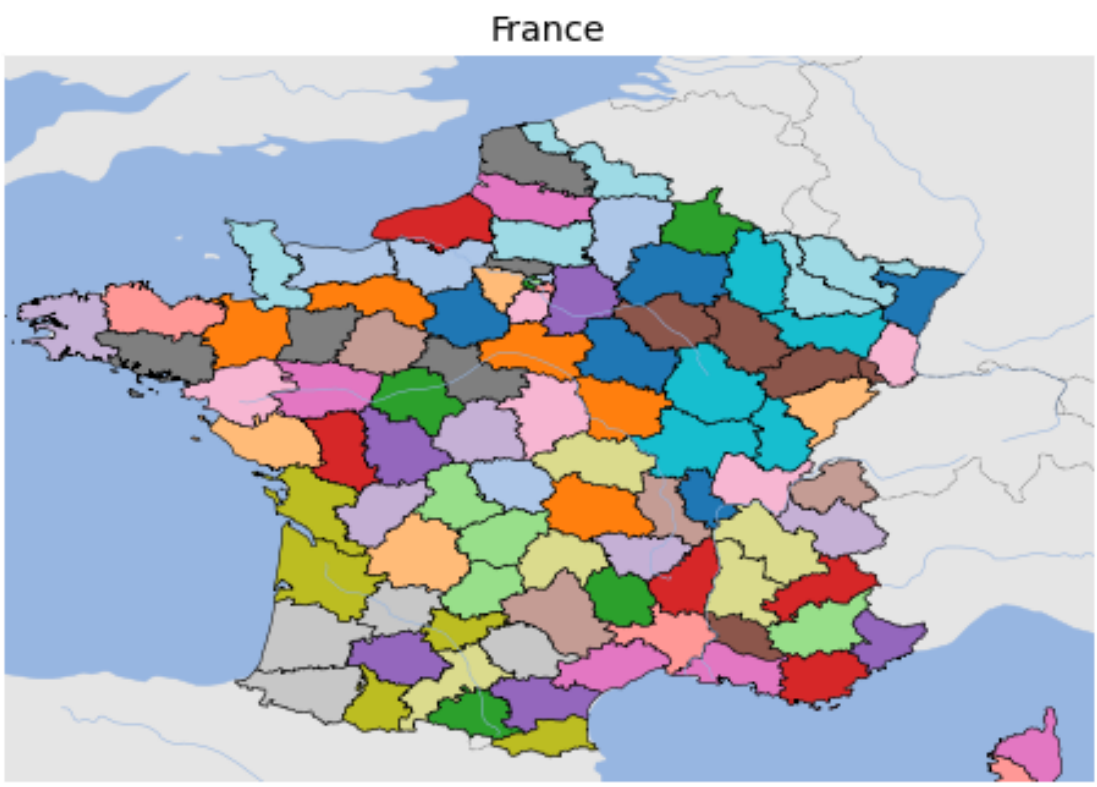
```
    sinPhiit0 = math.tanh(latiso + constantes.GRS80E * math.atanh(constantes.GRS80E *␣
↪math.sin(1)))
    sinPhiit1 = math.tanh(latiso + constantes.GRS80E * math.atanh(constantes.GRS80E *␣
↪sinPhiit0))
    sinPhiit2 = math.tanh(latiso + constantes.GRS80E * math.atanh(constantes.GRS80E *␣
↪sinPhiit1))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

[36]:
```
((-4.1615802638173065, 41.303505287589545),
 (10.699505053975292, 50.85243395553585))
```

[37]:
```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
ax = carte_france((8,8))

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

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

polys = []
for i, (record, shape) in enumerate(zip(records, shapes)):
    # les coordonnées sont en Lambert 93
    if i == 0:
        print(record, shape.parts)
    geo_points = [lambert932WGPS(x,y) for x, y in shape.points]
    if len(shape.parts) == 1:
        # Un seul polygone
        poly = Polygon(geo_points)
```

```
    else:
        # Il faut les fusionner.
        ind = list(shape.parts) + [len(shape.points)]
        pols = [Polygon(geo_points[ind[i]:ind[i+1]]) for i in range(0, len(shape.
 ↪parts))]
        try:
            poly = unary_union(pols)
        except Exception as e:
            print("Cannot merge: ", record)
            print([_.length for _ in pols], ind)
            poly = Polygon(geo_points)
    polys.append(poly)

data = geopandas.GeoDataFrame(geometry=polys)
# cmap -> voir https://matplotlib.org/users/colormaps.html
data.plot(ax=ax, cmap='tab20', edgecolor='black');
# Ou pour définir des couleurs spécifiques.
# geopandas.plotting.plot_polygon_collection(ax, data['geometry'], data['colors'],␣
 ↪values=None)
```

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



1.4.6 exercice 3 : résultats des élections par départements

Ce n'est pas toujours évident !

[38] :

```
```

1.5 seaborn

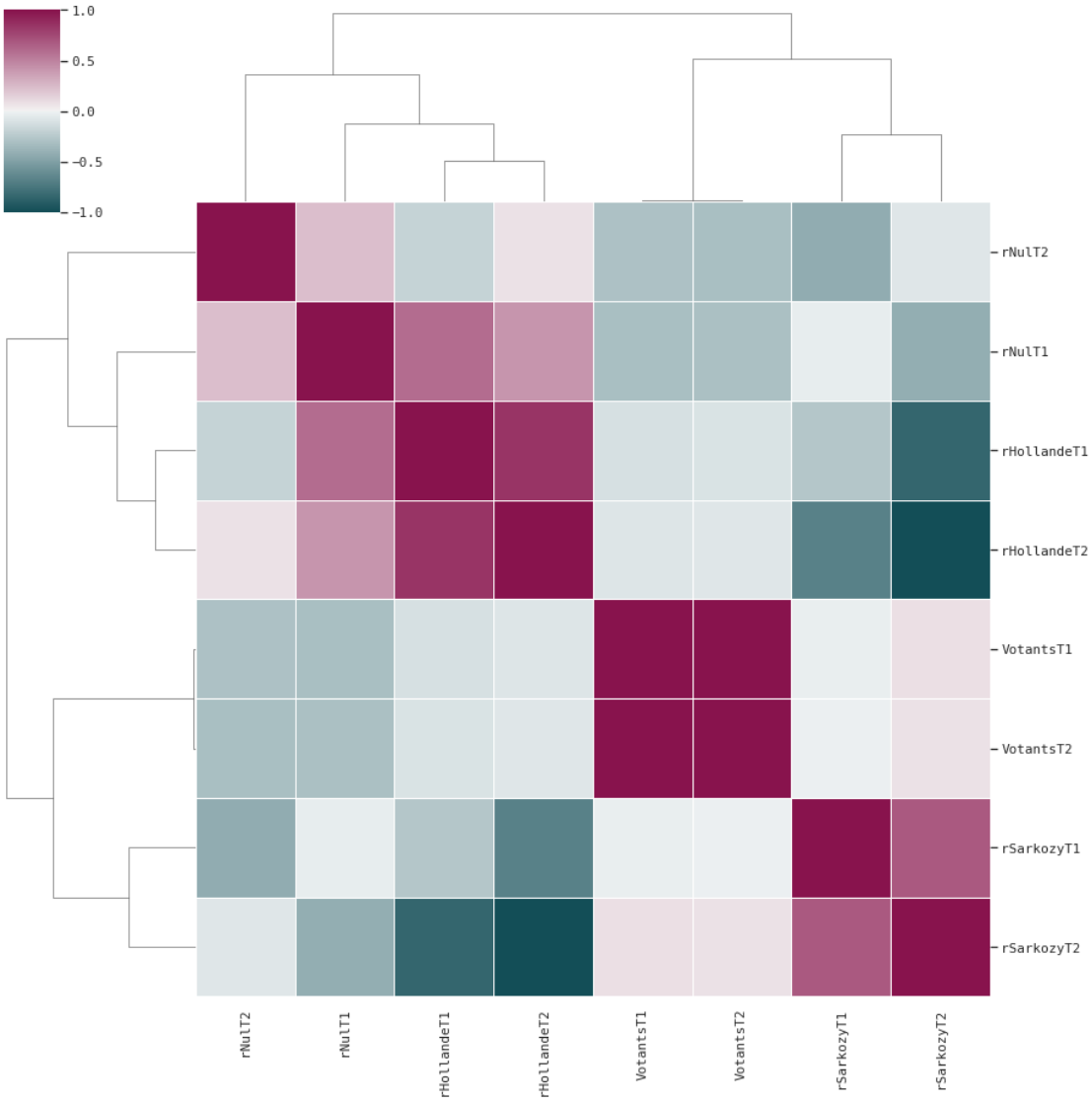
seaborn propose des graphiques inéressants pour un statisticien. La gallerie en donne un bon aperçu. On retravaille peu les graphiques. Le code suivant montre les corrélations entre variables pairplot.
warning les warnings sont la plupart du temps dûs au fait que *seaborn* a été testé sur une version antérieure d'une de ses dépendances comme *matplotlib* et qu'il n'est pas encore à jour pour tenir compte des derniers développement.

[39] :

```
import seaborn
seaborn.pairplot(data_elections);
```

Celui-ci est aussi intéressant : clustermap pour étudier les corrélations.

[40]:
```
import seaborn
seaborn.set(font="monospace")

cmap = seaborn.diverging_palette(h_neg=210, h_pos=350, s=90, l=30, as_cmap=True)

seaborn.clustermap(data_elections.corr(), linewidths=.5, figsize=(13, 13), cmap=cmap);
```



1.6 bokeh

bokeh propose des graphiques en javascript. La gallerie est moins fournie que celle de matplotib. Le principale avantage de bokeh est de proposer gaphiques zoomables (interactifs).

1.6.1 initialisation

La première étape est de signifier que la sortie se fera dans un notebook.

[41]:
```
from bokeh.plotting import output_notebook
output_notebook()
```

1.6.2 premier graphe

On utilise bokeh pour un simple graphique XY. on peut choisir les différentes options interactives, zoom … Voir tools.

[42]:
```
from bokeh.plotting import figure, show

p = figure(title = "élections")
p.title.text = "élections"
p.circle(data_elections["rHollandeT1"], data_elections["rHollandeT2"], color="red",␣
 ↪fill_alpha=0.2, size=10, legend="H")
p.circle(data_elections["rSarkozyT1"], data_elections["rSarkozyT2"], color="blue",␣
 ↪fill_alpha=0.2, size=10, legend="S")
p.line([0.2,0.7], [0.2,0.7], line_color="green", line_dash="dashed")
p.xaxis.axis_label = "tour 1"
p.yaxis.axis_label = "tour 2"
show(p)
```

```
BokehDeprecationWarning: 'legend' keyword is deprecated, use explicit
'legend_label', 'legend_field', or 'legend_group' keywords instead
BokehDeprecationWarning: 'legend' keyword is deprecated, use explicit
'legend_label', 'legend_field', or 'legend_group' keywords instead
```

1.6.3 ajouter du texte

Les différents éléments qu'on peut ajouter au graphe s'appelle des Glyphes. La documentation manque parfois de précision. Pour la fonction text, il faut préciser le texte à afficher sous forme de liste.

[43]:
```
from bokeh.plotting import figure, show

p = figure(title = "élections")
p.title.text = "élections"
p.circle(data_elections["rHollandeT1"], data_elections["rHollandeT2"], color="red",␣
 ↪fill_alpha=0.2, size=10, legend="H")
p.circle(data_elections["rSarkozyT1"], data_elections["rSarkozyT2"], color="blue",␣
 ↪fill_alpha=0.2, size=10, legend="S")
p.line([0.2,0.7], [0.2,0.7], line_color="green", line_dash="dashed")
p.xaxis.axis_label = "tour 1"
p.yaxis.axis_label = "tour 2"


def display_text(p, row):
    p.text(x=row["rHollandeT1"], y=row["rHollandeT2"], text=[row["Libellé du␣
 ↪départementT1"]],
           text_font_size="8pt", color="black", text_align="left",␣
 ↪text_baseline="middle", angle=0)
    return row["Libellé du départementT1"]
```

```
data_elections.apply(lambda row: display_text(p, row), axis=1)

show(p)
```

```
BokehDeprecationWarning: 'legend' keyword is deprecated, use explicit
'legend_label', 'legend_field', or 'legend_group' keywords instead
BokehDeprecationWarning: 'legend' keyword is deprecated, use explicit
'legend_label', 'legend_field', or 'legend_group' keywords instead
```

1.6.4 composition de graphes

Voir linked brushing.

[44]:
```
from bokeh.plotting import figure, show, gridplot
size = 400

ph = figure(title = "élections", width=size, height=size)
ph.title.text = "élections"
ph.circle(data_elections["rHollandeT1"], data_elections["rHollandeT2"], color="red",␣
 ↪fill_alpha=0.2, size=10, legend="H")
ph.line([0.2,0.7], [0.2,0.7], line_color="green", line_dash="dashed")
ph.xaxis.axis_label = "tour 1"
ph.yaxis.axis_label = "tour 2"

ps = figure(title = "élections", width=size, height=size)
ps.title.text = "élections"
ps.circle(data_elections["rSarkozyT1"], data_elections["rSarkozyT2"], color="blue",␣
 ↪fill_alpha=0.2, size=10, legend="S")
ps.line([0.2,0.7], [0.2,0.7], line_color="green", line_dash="dashed")
ps.xaxis.axis_label = "tour 1"
ps.yaxis.axis_label = "tour 2"

p = gridplot([[ph, ps]])

show(p)
```

```
BokehDeprecationWarning: 'legend' keyword is deprecated, use explicit
'legend_label', 'legend_field', or 'legend_group' keywords instead
BokehDeprecationWarning: 'legend' keyword is deprecated, use explicit
'legend_label', 'legend_field', or 'legend_group' keywords instead
```

1.6.5 interactions avec bokeh

Tout est expliqué dans la page interaction.

[45]:
```
from bokeh.models.widgets import Panel, Tabs
from bokeh.io import show
from bokeh.plotting import figure

ph = figure(title="élections")
ph.circle(data_elections["rHollandeT1"], data_elections["rHollandeT2"], color="red",␣
 ↪fill_alpha=0.2, size=10, legend="H")
ph.line([0.2,0.7], [0.2,0.7], line_color="green", line_dash="dashed")
ph.xaxis.axis_label = "tour 1"
```

```
ph.yaxis.axis_label = "tour 2"

ps = figure(title = "élections")
ps.title.text = "élections"
ps.circle(data_elections["rSarkozyT1"], data_elections["rSarkozyT2"], color="blue",␣
↪fill_alpha=0.2, size=10, legend="S")
ps.line([0.2,0.7], [0.2,0.7], line_color="green", line_dash="dashed")
ps.xaxis.axis_label = "tour 1"
ps.yaxis.axis_label = "tour 2"

tab1 = Panel(child=ph, title="Hollande")
tab2 = Panel(child=ps, title="Sarkozy")

tabs = Tabs(tabs=[ tab1, tab2 ])

show(tabs)
```

```
BokehDeprecationWarning: 'legend' keyword is deprecated, use explicit
'legend_label', 'legend_field', or 'legend_group' keywords instead
BokehDeprecationWarning: 'legend' keyword is deprecated, use explicit
'legend_label', 'legend_field', or 'legend_group' keywords instead
```

Si on connaît un peu de javascript, on peut créer des graphes qui peuvent interagir avec la souris à n'importe quel endroit du graphe.

1.7 interaction avec matplotlib (ou bokeh)

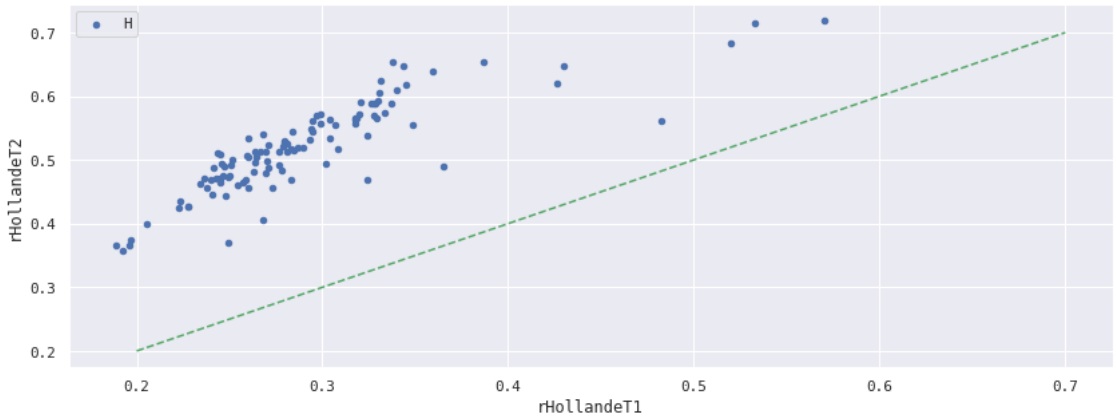
1.7.1 dropdown

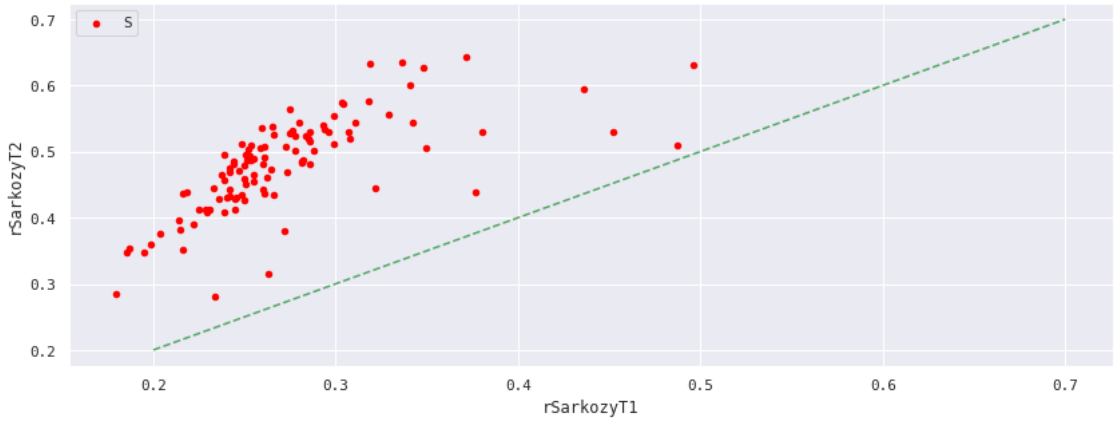
Le module interact permet de changer le contenu d'une cellule en fonction d'un bouton, d'une barre de défilement… Le code de cette fonction est sur github. La liste des widget possible est dans le notebook WidgetList

[46]:
```
import matplotlib.pyplot as plt
from IPython.html.widgets import interact, Dropdown

def plot(cand):

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

cand = Dropdown(options=['Hollande', 'Sarkozy'], value='Hollande',␣
↪description='candidat')

interact(plot, cand=cand)
print("")
```

```
C:\Python395_x64\lib\site-packages\IPython\html.py:12: ShimWarning: The
`IPython.html` package has been deprecated since IPython 4.0. You should import
from `notebook` instead. `IPython.html.widgets` has moved to `ipywidgets`.
  warn("The `IPython.html` package has been deprecated since IPython 4.0. "
```

```
interactive(children=(Dropdown(description='candidat', options=('Hollande', 'Sarkozy'),␣
↪value='Hollande'), Out...
```





Ces interactions ne sont pas limitées à matplotlib. Tout type de sortie peut dépendre d'un *widget*.

1.7.2 exercice 4 : même code, widget différent

[47] :

```
```

1.8 autres options

Il existe un grande nombre de modules permettant de dessiner. Les plus récents utilisent le javascript.

- static

  - ggplot : aspect similare à matplotlib version ggplot mais la syntaxe est différente

- Javascript

  - pygal : voir exemple plus bas, aspect réussi, le module prévoit une extension pour les carte, le résultat nécessite l'ajout de quelques scripts Javascript
  - mpld3 : créer un graphe avec la syntaxe Javascript et le transformer en Javascript
  - folium : Javascript + OpenStreetMap

[48]:
```
html_pygal = """
<script type="text/javascript" src="http://kozea.github.com/pygal.js/javascripts/svg.
 ↪jquery.js"></script>
<script type="text/javascript" src="http://kozea.github.com/pygal.js/javascripts/
 ↪pygal-tooltips.js"></script>
{pygal_render}
"""

from IPython.core.display import HTML
import pygal
xy_chart = pygal.XY(stroke=False)
xy_chart.title = 'Elections 2012'
xy = list(zip(data_elections["rHollandeT1"], data_elections["rHollandeT2"]))
xy_chart.add('H', xy)
HTML(html_pygal.format(pygal_render=xy_chart.render().decode()))
```

[48]:
```
<IPython.core.display.HTML object>
```

[49]:
```

```