académie Créteil

RÉGION ACADÉMIQUE
ÎLE-DE-FRANCE

SFdS

Société Française de Statistique

STATISTIQUE, PROBABILITÉS ET JUGEMENT CRITIQUE

Journées nationales de l'APMEP
BORDEAUX
21-22 octobre 2018

STATISTIQUE, PROBABILITÉS ET JUGEMENT CRITIQUE

Joël BERHOUET
Professeur au lycée Pablo PICASSO – Fontenay-sous-Bois

Philippe DUTARTE
IA-IPR de mathématiques – Académie de Créteil

Fabienne GLEBA
Professeure au collège De Lattre – Le Perreux-sur-Marne

SOMMAIRE

I. Un regard critique sur les représentations graphiques…

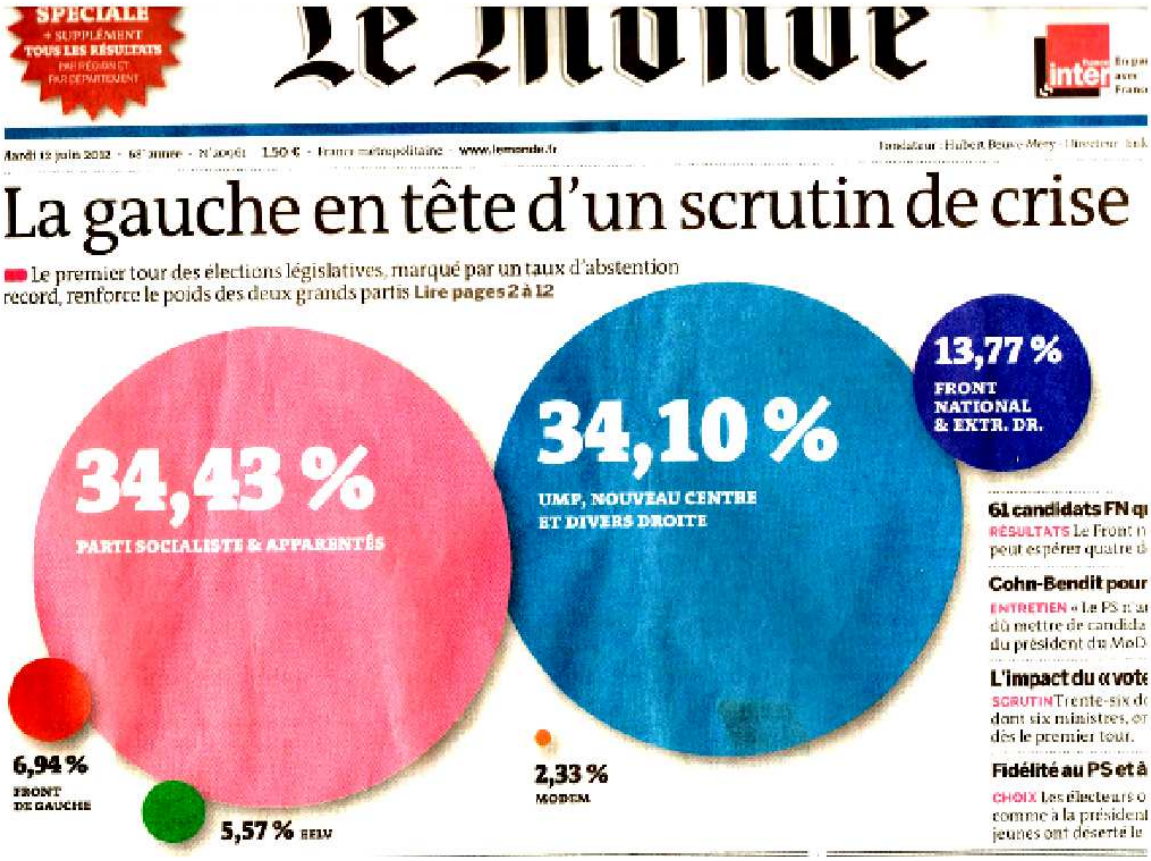
1. Éducation civique et aire du disque

(niveau collège)

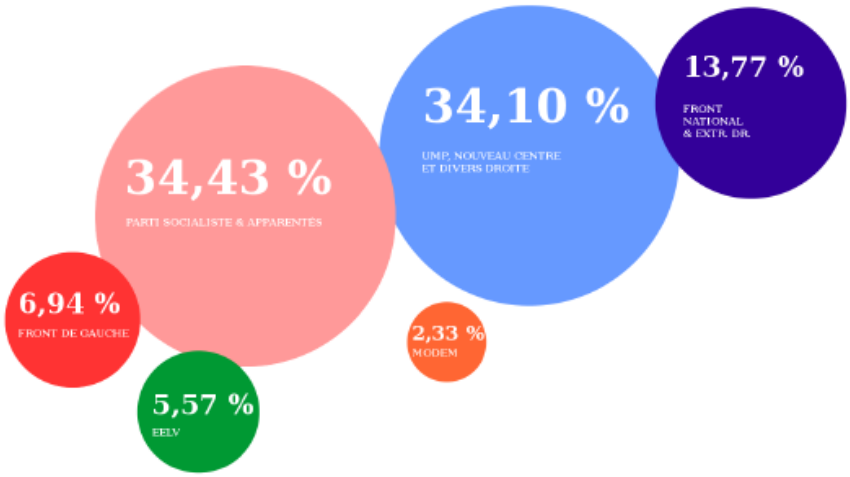
Des habitudes médiatiques fréquentes conduisent à comparer des données en les représentant par des disques ; cette vision dans le plan est plus expressive que des tableaux ou des diagrammes en bâtons. Encore faut-il s'appuyer sur la bonne intuition qui y est liée. Le bon sens conduit à voir les valeurs des données comme **proportionnelles aux aires** des disques qui leur sont associés.

L'image ci-dessous présente une figure en première page du journal « *Le Monde* » paru le lundi 11 mai 2012, lendemain du premier tour d'élections législatives, illustrant les pourcentages de votants (par rapport aux suffrages exprimés) des familles politiques regroupées en 6 catégories :

– Parti socialiste et apparentés 34,43% ;
– UMP, nouveau centre et divers droite 34,10% ;
– Front national et extrême droite 13,77% ;
– Front de gauche 6,94% ;
– Europe Ecologie Les Verts 5,57% ;
– MODEM 2,33% ;



On présente maintenant une « bonne représentation » :

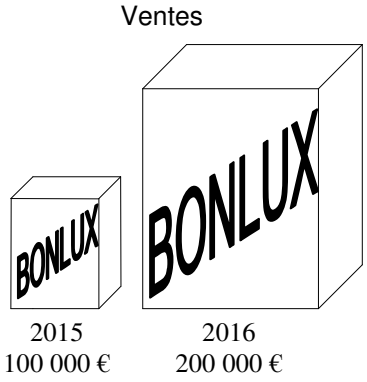


Commentez.

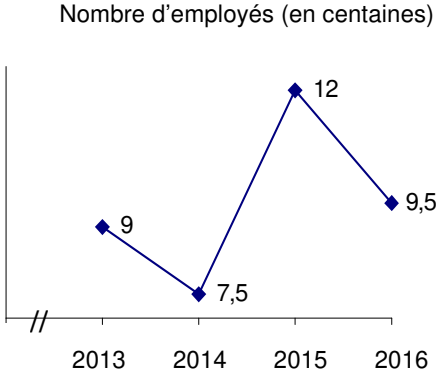
2. Diagrammes trompeurs ou critiquables

(à partir du collège)

1. Expliquer pourquoi le diagramme suivant est incorrect.



2. Expliquer pourquoi le diagramme suivant est critiquable.



3. Chômage

(à partir du collège)

Le tableau suivant donne le nombre de chômeurs en France métropolitaine pour l'année 2016 (au sens du bureau international du travail, source INSEE).

2016	Nombre de chômeurs (milliers)
trimestre 1	2 852
trimestre 2	2 788
trimestre 3	2 798
trimestre 4	2 790

Préparer deux graphiques :
a. l'un destiné à illustrer un article intitulé « le chômage demeure important » ;
b. l'autre destiné à illustrer un article intitulé « le chômage baisse ».

4. D'autres exemples

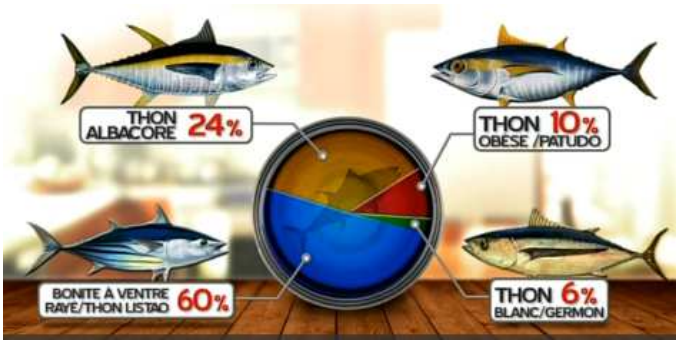
On pourra consulter les adresses suivantes et, en particulier, les documents proposés par Nicolas Gauvrit :
http://cortecs.org/materiel/mathematiques-comment-tromper-avec-des-graphiques/
http://images.math.cnrs.fr/Graphiques-frelates.html
http://images.math.cnrs.fr/+Graphique-trompeur

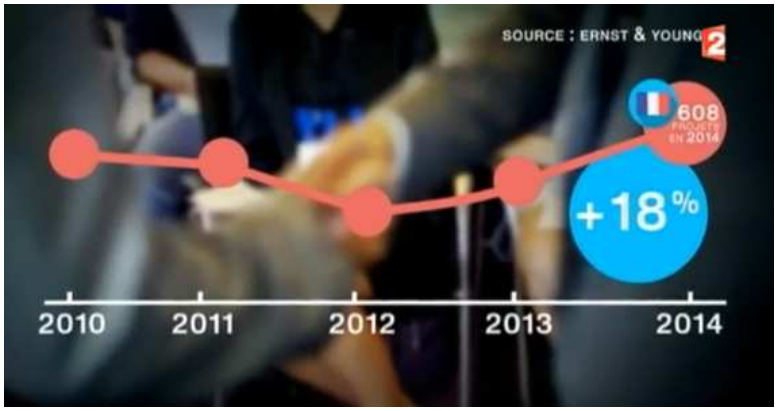
Journal Télévisé – 20h – *TF1*

Journal Télévisé – 20h – *France 2*

Journal Télévisé – 20h – *France 2*

Des ressources vidéo sur le site mathix des « Dudu » (niveau collège) :
https://mathix.org/linux/problemes-ouverts/autres-videos-images
https://mathix.org/video/problemes_ouverts/recherche/index.php





Comment faire un mauvais graphique ?

Howard Wainer, statisticien américain, a établi en 1984 des règles pour réaliser un mauvais graphique[1] ! Il faut donc éviter de les appliquer…
Howard Wainer – *How to Display Data Badly* – *The American Statistician* (1984), article consultable à l'adresse suivante :
http://www.rci.rutgers.edu/~roos/Courses/grstat502/wainer.pdf

1. Moins les données apparaissent sur le graphique, mieux c'est.
2. Cachez au maximum les données sur le graphique.
3. Ne faites pas attention à respecter visuellement les données.
4. Seul l'ordre importe.
5. Ne tenez pas compte du contexte.
6. Changez d'échelle au milieu des axes.
7. Insistez sur ce qui est trivial, ignorez l'important.
8. Mettez ensemble des quantités d'ordre de grandeur très différent.
9. L'ordre alphabétique est primordial.
10. N'hésitez pas à utiliser des légendes illisibles, incomplètes, incorrectes et même ambiguës.
11. Utilisez le maximum de décimales et de dimensions.
12. Ce n'est pas parce que vous avez bien travaillé dans le passé, qu'il faut continuer à le faire.

Une illustration de la règle 3 :

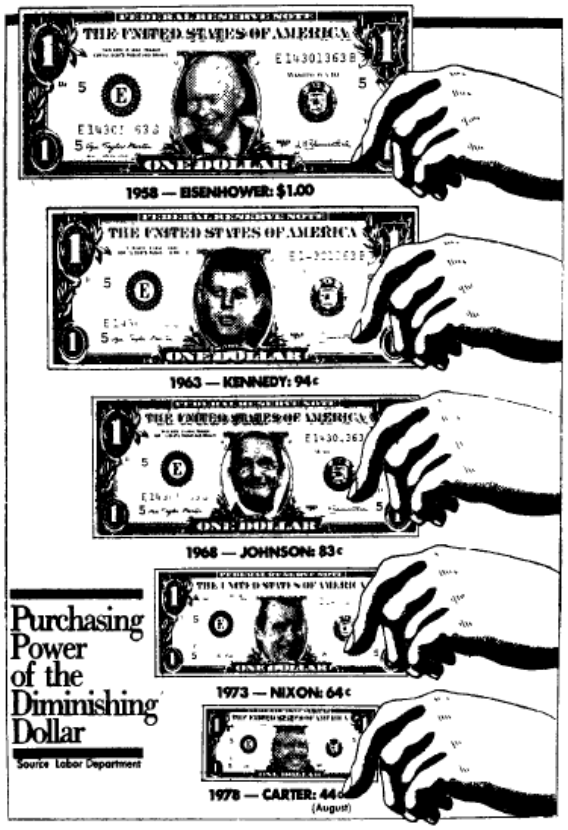


Figure 9. An example of how to goose up the effect by squaring the eyeball (© 1978, The Washington Post)

[1] Cité par Jean-Jacques Droesbeke, Catherine Vermandele – *Les nombres au quotidien, leur histoire, leurs usages* Technip 2016.

II. Peut-on croire un sondage ?

5. Abaque de calcul de la marge d'erreur et simulations

La théorie statistique permet de mesurer l'incertitude à attacher à chaque résultat d'une enquête. Cette incertitude s'exprime par un intervalle de confiance situé de part et d'autre de la valeur observée et dans lequel la vraie valeur a une probabilité déterminée de se trouver. Cette incertitude, communément appelée « marge d'erreur », varie en fonction de la taille de l'échantillon et du pourcentage observé comme le montre le tableau ci-dessous :

INTERVALLE DE CONFIANCE A 95% DE CHANCE

Et si l'effectif est...	Si le pourcentage trouvé est... 5 ou 95%	10 ou 90%	20 ou 80%	30 ou 70%	40 ou 60%	50%
50	6,2	8,5	11,3	13,0	13,9	14,1
100	4,4	6,0	8,0	9,2	9,8	10,0
200	3,1	4,2	5,7	6,5	6,9	7,1
250	2,8	3,8	5,1	5,8	6,2	6,3
300	2,5	3,5	4,6	5,3	5,7	5,8
350	2,3	3,2	4,3	4,9	5,2	5,3
400	2,2	3,0	4,0	4,6	4,9	5,0
450	2,1	2,8	3,8	4,3	4,6	4,7
500	1,9	2,7	3,6	4,1	4,4	4,5
600	1,8	2,4	3,3	3,7	4,0	4,1
700	1,6	2,3	3,0	3,5	3,7	3,8
800	1,5	2,1	2,8	3,2	3,5	3,5
900	1,4	2,0	2,6	3,0	3,2	3,3
1000	1,4	1,8	2,5	2,8	3,0	3,1
2000	1,0	1,3	1,8	2,1	2,2	2,2
4000	0,7	0,9	1,3	1,5	1,6	1,6
6000	0,6	0,8	1,1	1,3	1,4	1,4
10000	0,4	0,6	0,8	0,9	0,9	1,0

Exemple de lecture du tableau : dans le cas d'un échantillon de 1000 personnes, si le pourcentage mesuré est de 10%, la marge d'erreur est égale à 1,8. Le vrai pourcentage est donc compris entre 8,2% et 11,8%.

Source : Ifop (Institut Français d'Opinion Publique).

Enoncé

On note p la proportion inconnue d'un caractère dans une population. On souhaite estimer cette proportion p par une fourchette de sondage, c'est-à-dire un intervalle de la forme $[f - marge\ ,\ f + marge]$ où :

- f désigne la fréquence du caractère dans un échantillon de taille n ;
- *marge* est la marge d'erreur lue dans le tableau de l'Ifop.

1. Compléter la fonction *echantillon* ci-dessous afin qu'elle simule, sur un échantillon de taille n, la fréquence d'un caractère dont la proportion dans la population est égale à p.

```
import random

def echantillon(n,p):
    s = 0
    for k in range(n):
        if random.random() < p:
            s = ......
    return s / n
```

2. Utiliser cette fonction afin d'obtenir une fréquence f dans le cas d'un échantillon de taille $n = 1000$ et avec une proportion p (a priori inconnue) égale à 0,2.
En déduire la fourchette de sondage correspondante. Contient-elle p ?

3. Compléter la fonction *test_fourchette* afin qu'elle renvoie 1 si la proportion p que l'on cherche à estimer est contenue dans la fourchette de sondage et 0 sinon. (La marge est lue dans le tableau en considérant que p et f sont proches).

```
def test_fourchette(n,p,marge):
    f = echantillon(n,p)
    if ... and ... :
        return ...
    else:
        return ...
```

4. Tester cette fonction avec $n = 1\,000$ et $p = 0{,}2$. La fourchette contient-elle toujours la proportion p qu'elle cherche à estimer ?

5. Que fait la fonction ci-dessous ? Tester la à plusieurs reprises. Que peut-on dire de la valeur retournée ?

```
def mille_fourchettes(n,p,marge):
    s = 0
    for k in range(1000):
        s = s + test_fourchette(n,p,marge)
    return s / 1000
```

6. Que pensez-vous la phrase explicative qui suit le tableau ? Expliquez le titre « Intervalle de confiance à 95% de chance ».

7. Voici un sondage publié au Canada. Expliquez l'expression « 19 fois sur 20 ».

Le Canada est-il sur la bonne ou la mauvaise voie?

Le sondage, réalisé en collaboration avec iPolitics, a été effectué au téléphone et par Internet du 10 au 15 octobre 2014 auprès de 1671 Canadiens de 18 ans et plus. Sa marge d'erreur est de plus ou moins 2,4 points de pourcentage, 19 fois sur 20. .

Remarques :
– On peut simuler les fourchettes de sondages avec un tableur.
– En Terminale, on peut indiquer que la marge d'erreur est donnée par la formule suivante : $1{,}96\sqrt{\frac{f(1-f)}{n}}$.

III. La statistique inférentielle pour débusquer les charlatans

6. Sourcier or not sourcier ?

(d'après G. Charpak et H. Broch – *Devenez sorcier, devenez savant*).
(niveau lycée)

Illustration wikipedia

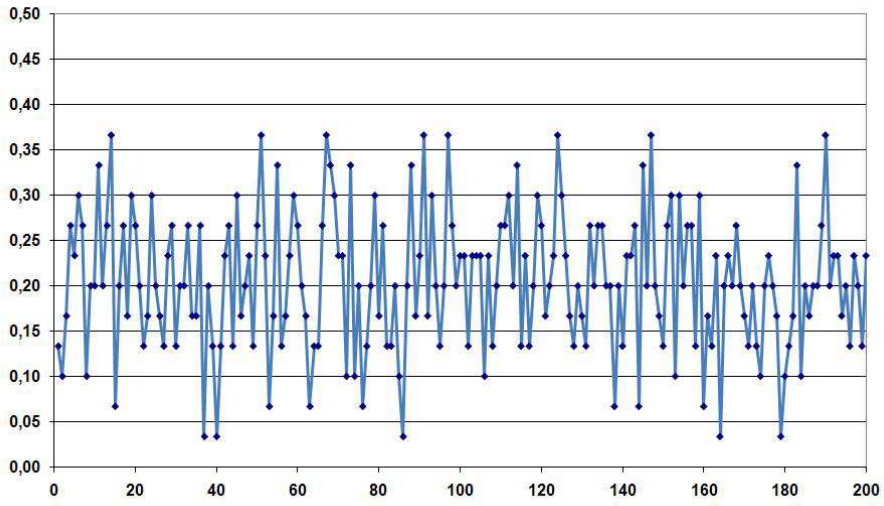
Un « sourcier » prétend posséder des pouvoirs lui permettant de détecter la présence d'eau à l'aide d'une baguette en bois. On met en place un dispositif permettant de tester les prétendus pouvoirs du sourcier. Cinq canalisations sont masquées dont une seule contient (aléatoirement) de l'eau. Le soucier doit désigner la canalisation contenant de l'eau.

Énoncé niveau seconde

1. Si le sourcier répond « au hasard », quelle probabilité p a-t-il de répondre correctement ?

2. Comme le sourcier ne prétend pas être infaillible, on fera 30 fois l'expérience. Si le sourcier répond au hasard, nous serons en présence d'un échantillon aléatoire de réponses, de taille 30, extrait d'une population où la fréquence de bonnes réponses est 0,2.

Le graphique ci-dessous montre les fluctuations des fréquences de bonnes réponses sur de tels échantillons.



a. Combien d'échantillons de taille 30 ont été simulés ?
b. D'après le graphique, est-il rare, en répondant au hasard, d'obtenir au moins 25 % de bonnes réponses ?
c. Peut-on, en répondant au hasard, obtenir 40 % de bonnes réponses ? Si oui, est-ce rare ?
d. D'après le programme de la classe de seconde générale et technologique, lorsqu'on prélève un échantillon de taille n d'une population où la proportion d'un caractère est p, alors, sous certaines conditions ici vérifiées, la probabilité que cet échantillon fournisse une fréquence appartenant à l'intervalle $\left[p-\frac{1}{\sqrt{n}},\ p+\frac{1}{\sqrt{n}}\right]$ est d'environ 0,95.
Que nous dit ce résultat appliqué à la situation présente ?
3. Le sourcier, sur les 30 expériences pratiquées, a obtenu 8 bonnes réponses. Doit-on penser qu'il possède un don ? Justifier.

Énoncé niveau première

1. Si le sourcier répond « au hasard », quelle probabilité p a-t-il de répondre correctement ?
2. Comme le sourcier ne prétend pas être infaillible, on fera 30 fois l'expérience. On suppose que le sourcier répond au hasard et on désigne par X la variable aléatoire correspondant au nombre de bonnes réponses qu'il fournit.
Montrer que X suit une loi binomiale dont on précisera les paramètres.
3. Le sourcier, sur les 30 expériences pratiquées, a obtenu 8 bonnes réponses. Doit-on penser qu'il possède un don ? Justifier.

7. Le riz d'Emoto

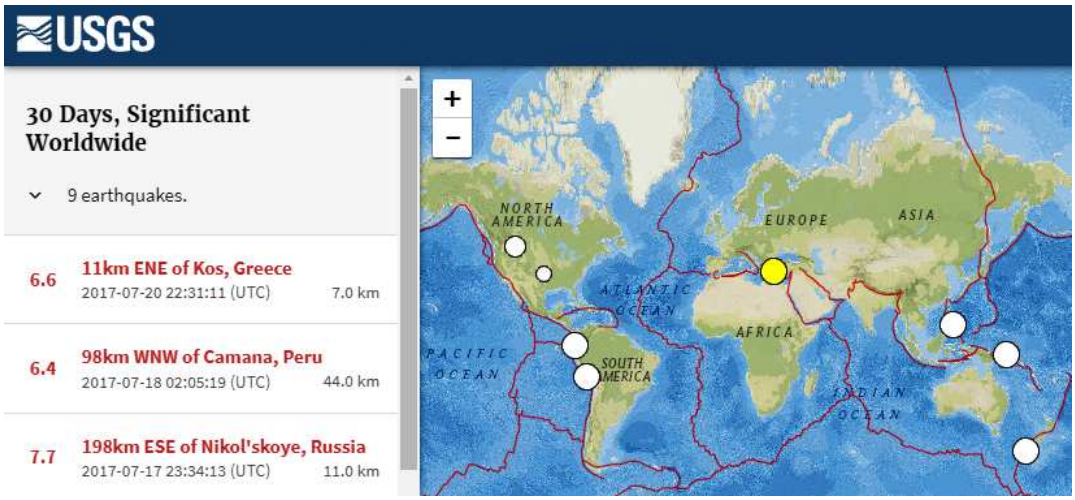
(niveau seconde)

En surfant sur le net, Cécile et Julien sont tombés sur la théorie du Dr Emoto qui prétend que l'on peut influencer la conservation des aliments par des pensées positives.
Pour tester cette théorie, ils réalisent le protocole suivant :
ils versent du riz cuit dans 100 pots et les associent deux par deux, formant ainsi 50 couples de pots numérotés 1A-1B, 2A-2B, …, 50A-50B ;
chaque matin, Cécile dit des mots doux aux pots portant la lettre A alors que Julien, à contrecœur, dit des méchancetés aux pots portant la lettre B ;
après 15 jours, ils comparent l'état de conservation des pots dans chacun des couples.
A la fin de leur expérience, ils ont observé que les pots A étaient mieux conservés dans 30 couples parmi les 50.
On note E l'événement « dans un couple, le pot A est mieux conservé que le pot B ».
1. Quelle est la fréquence f de l'événement E dans l'échantillon étudié par Cécile et Julien ?
2. A quelle probabilité p faut-il comparer la fréquence f ? A quelle hypothèse correspond cette probabilité ?
3. Y-a-t-il une différence « statistiquement significative » entre f et p ? Justifiez à l'aide de l'intervalle de fluctuation au seuil de 95%.
4. Ce résultat va-t-il dans le sens de la théorie d'Emoto ?
http://www.scilogs.fr/raisonetpsychologie/le-flegme-troublant-du-riz-thai/
http://blogs.univ-poitiers.fr/n-yeganefar/2014/05/13/quand-les-scientifiques-samusent-operation-emoto-riz/

8. Pseudo-sciences, sismologie et coïncidences

(niveau seconde)

La situation suivante est inspirée de l'ouvrage *Devenez sorciers, devenez savants*, d'Henri Broch et Georges Charpak. Imaginons un individu se prétendant astrologue et faisant des prévisions pour l'avenir. Pour les années 2014, 2015, 2016 (cette dernière année étant bissextile), il avait donné une liste de 129 jours sismiques dans le monde (c'est-à-dire avec au moins un tremblement de Terre de magnitude supérieure ou égale à 6,5).



La consultation du site du *United States Geological Survey* nous indique que sur cette période, ont été recensés 158 tremblements de Terre majeurs sur 134 jours différents, dont 19 coïncident avec les prévisions du « voyant ». Ce dernier fait-il significativement mieux que le hasard ?

1. Déterminer la proportion p des jours sismiques durant les trois années 2014, 2015 et 2016.
2. On prélève un échantillon au hasard de 129 dates durant ces trois années (on suppose qu'il s'agit d'un prélèvement avec remise), donner l'intervalle de fluctuation au seuil de 95 % de la fréquence des jours sismiques sur un tel échantillon.
3. Situer la fréquence f obtenue par l'astrologue par rapport à l'intervalle précédent et conclure.
4. Autre approche : estimer la probabilité de faire mieux que l'astrologue avec un choix au hasard à l'aide du programme JoursSismiques.py.

IV. Probabilités, algorithmique et coïncidences

9. Psychogénéalogie et coïncidences

(niveau seconde)

Dans les années 1970, la psychologue Anne Ancelin Schützenberger développa une théorie d'inspiration psychanalytique nommée «psychogénéalogie». Selon cette théorie, un inconscient familial travaille si bien dans l'ombre qu'on peut contracter des maladies ou des troubles mentaux à certaines dates parce qu'un de nos ancêtres aurait lui aussi vécu quelque chose de remarquable à cette date. Disons tout de suite que l'idée présentée comme cela n'est pas absurde : on peut imaginer que quelqu'un commence une dépression le jour anniversaire de la mort de ses parents, par exemple. En revanche dans la théorie de Schützenberger, il peut s'agir de cas bien plus mystérieux. Ainsi, elle imagine qu'on peut déclarer un cancer le jour anniversaire de l'accident d'un grand-oncle, et cela même si nous ne savons pas qu'un tel accident a eu lieu.
L'argument massue de la psychogénéalogie est nommé le « syndrome des anniversaires » : Schützenberger a en effet noté que, si l'on cherche bien, on finit souvent par retrouver des coïncidences de dates, bref des anniversaires communs entre événements.
La thérapie psychogénéalogique consiste à rechercher au moyen d'une enquête généalogique les dates importantes concernant nos ancêtres (naissance, majorité, premier amour, maladie, accident, mort, etc.), en remontant aussi loin qu'il le faut pour qu'une de ces dates se trouve être celle d'un événement important pour nous (accident, début d'une dépression, déclaration d'une maladie, etc.). Le nombre de dates recueillies lors de l'enquête dépasse bien souvent la cinquantaine et parfois la centaine, ce qui laisse planer un doute sur la nature improbable des coïncidences. […]
La question qu'on doit se poser est celle-ci : si nous prenons deux listes de dates (disons n et m dates), quelle est la probabilité qu'une date de la première liste soit la même qu'une date de la seconde ? Avec une centaine de dates concernant les ancêtres, et une dizaine nous concernant, la probabilité de collision est alors de 0,96 environ. Que deux dates coïncident est en réalité beaucoup moins étonnant que l'événement inverse.
Jean-Paul Delahaye, Nicolas Gauvrit – *Comme par hasard !* book-e-book 2012.

1. Implanter les fonctions suivantes sur Python.

```
import random
import matplotlib.pyplot as plt

def liste_dates(n):
    # Liste aleatoire de n dates sans remise
    dates = random.sample(range(1,366),n)
    return dates

def coincidence(n, m):
    # Recherche d'au moins une coincidence entre deux listes de n dates et m dates
    dates_moi = liste_dates(n)
    # print(dates_moi)
    dates_ancetres = liste_dates(m)
    # print(dates_ancetres)
    double = 0
    for d in dates_moi:
        if d in dates_ancetres:
            #print(d)
            double = 1
    return double

def EvolutionFrequence(n, m, N):
    # Evolution de la frequence d'au moins une coincidence entre deux listes de tailles n et m pour N repetitions de l'expérience
    s = 0
    for k in range(1, N + 1):
        s = s + coincidence(n, m)
        plt.plot(k, s/k, 'b.')
    plt.grid()
    plt.show()
```

2. Effectuer quelques expériences de recherche de coïncidences entre deux listes aléatoires de 10 dates et de 100 dates, en imprimant les listes.
Qu'observe-t-on ?
3. Représenter l'évolution de la fréquence de l'événement E : « il existe au moins une coïncidence entre une liste aléatoire de 10 dates et une liste aléatoire de 100 dates » lorsque l'on répète l'expérience du choix aléatoire des listes de dates.
Vérifie-t-on l'affirmation du texte ?
4. Combien de fois suffit-il de répéter l'expérience pour obtenir une estimation de la probabilité de E à 10^{-2} près au seuil de 95 % ? (On admet que compte-tenu de la fluctuation d'échantillonnage, la fréquence obtenue après la répétition de n expériences fournit dans environ 95 % des cas une estimation de la probabilité de E à $\frac{1}{\sqrt{n}}$ près.)

Il n'y a pas de hasard, il n'y a que des rendez-vous

« Il n'y a pas de hasard, il n'y a que des rendez-vous ». Cette belle pensée est attribuée à Paul Eluard, sans qu'aucune recherche Internet n'en fournisse la source (jusqu'à preuve du contraire). Mais elle plait beaucoup aux mystiques de tout poil et pourrait être le ressort de bien des théories complotistes.
Le magazine *Books*, dans son numéro 85 « *Hasard et destin* » de septembre-octobre 2017, cite l'extrait suivant de l'ouvrage du psychologue américain Keith E. Stanovich, *How to Think Straight About Psychology* (Pour avoir les idées claires sur la psychologie), Pearson 2012 (10e édition).

Une célèbre liste pointe des coïncidences « troublantes » entre les présidents Abraham Lincoln et John F. Kennedy :
1. Lincoln a été élu en 1860 ; Kennedy en 1960.
2. Lincoln et Kennedy étaient tous deux attachés aux droits civisues.
3. Les noms de Lincoln et Kennedy ont sept lettres.
4. Lincoln avait une secrétaire nommée Kennedy et Kennedy une secrétaire nommée Lincoln.
5. Ils ont tous deux un successeur nommé Johnson.
6. Leurs assassins étaient connus par leurs trois noms (John Wilkes Booth et Lee Harwey Oswald).
7. Booth et Oswald adhéraient à des idées politiques impopulaires.
8. Booth a tiré sur Lincoln dans un théâtre et s'est caché dans un entrepôt ; Oswald a tiré sur Kennedy dans un entrepôt et s'est caché dans un théâtre (plus précisément une salle de cinéma).
Mais, comme John Leavy, spécialiste de la programmation informatique à l'Université du Texas l'a démontré dans un article de 1992, on peut établir le même genre de liste pour la plupart des présidents pris deux à deux.

V. Big data et citoyenneté

Chaque jour, nous générons 2,5 milliards de milliards d'octets de données et plus de 90 % des données existantes ont été créées ces deux dernières années. L'exploitation de ces « big data », dont le potentiel économique est gigantesque, nécessite des techniques statistiques, mathématiques et informatiques en pleine évolution constituant un pôle important de recherche lié à la notion d'intelligence artificielle. Cependant plusieurs types de risques d'atteinte à la vie privée ou aux droits fondamentaux sont cités, notamment après les révélations d'Edward Snowden en 2013. Ainsi, environ 80 % des données personnelles mondiales seraient détenues par les GAFA (Google, Apple, Facebook, Amazon). On comprend le sentiment de défiance que peuvent provoquer ces technologies à l'égard des algorithmes et de l'intelligence artificielle comme en témoignent ces affiches photographiées à Londres.

Dans les rues de Londres en 2018…

Une attitude plus constructive consiste plutôt à renforcer l'éducation des futurs citoyens en matière d'analyse des données massives, permettant ainsi de mieux voir et appréhender le monde dans lequel on vit, comme dans l'exemple des locations Airbnb ou dans celui des entreprises et des salaires en France, voire de participer au contrôle de la vie citoyenne, comme dans le cas de l'analyse des « Paradise papers ». Un aperçu d'exploitation en classe de lycée de ces exemples est donné ici à l'aide du module Pandas de Python.

10. Airbnb

Le site insideairbnb.com[2] pose la question suivante : « Comment Airbnb est-il réellement utilisé et affecte-t-il les quartiers de votre ville ? ». Pour répondre à cette question, il est possible d'y télécharger les données Airbnb de nombreuses villes dans le monde dont Paris. On obtient pour Paris (avril 2017) un fichier csv de 56 450 lignes, correspondant chacune à une location, pour 12 variables étudiées, dont le prix, la disponibilité, le nombre d'avis et les coordonnées géographiques.
On propose ici, à partir du fichier Airbnb_Paris.csv, un « data challenge » pour des élèves de lycée.

Data challenge « Airbnb »
Data scientist : le métier le plus attrayant du XXI^{e} siècle ? Le terme de "data scientist" à été inventé par Dhanurjay Patil (LinkedIn) et Jeff Hammerbacher (Facebook) en cherchant comment caractériser les métiers des données pour afficher des offres d'emploi :
« Analyste, ça fait trop Wall Street ; statisticien, ça agace les économistes ; chercheur scientifique, ça fait trop académique. Pourquoi pas "data scientist" ? ».

Explorer, seul ou en groupe, à l'aide de la bibliothèque Pandas de Python, le fichier de données Airbnb_Paris.csv en suivant la méthode suivante :
1. Lister les variables et déterminer leur type.
2. Rechercher les données manquantes ou les valeurs aberrantes.
3. Calculer des indicateurs statistiques, effectuer des groupements de données, visualiser.
4. Interpréter.

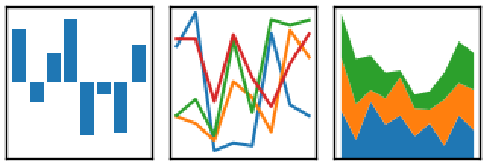
Que les meilleurs data scientists gagnent !

[2] Ce site a été crée par Murray Cox, écrivain et informaticien indépendant se qualifiant de « data activiste ».

Mémento Pandas

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



Installation de Pandas et lecture d'un fichier csv	
Installer Pandas : saisir dans la console l'une des instructions ci-contre.	`conda install pandas` `pip install pandas`
Importer le module Pandas	`import pandas`
Lire le fichier nom.csv figurant dans le même dossier que le programme Python et affecter les données à un « dataframe » (table de données dont chaque colonne correspond à une variable statistique) nommé ici df. Les tableaux de données manipulés par Pandas sont des dataframe.	`df = pandas.read_csv('nom.csv', sep = ';')` `sep = ','` si le séparateur du fichier csv est une virgule. `sep = '\t'` si le séparateur du fichier csv est une tabulation. Ajouter l'argument `encoding = 'latin-1'` si le fichier csv comporte des accents. Ajouter l'argument `decimal = ','` si les nombres décimaux du fichier csv sont séparés par une virgule.
Opérations de base sur un dataframe et ses colonnes	
Dimension du dataframe df : nombre de lignes et de colonnes (c'est-à-dire de variables).	`df.shape`
(Nom) et type des variables (ou colonnes).	`df.dtype`
Afficher les 5 premières lignes de df.	`df.head(5)`
Afficher les 5 dernières lignes de df.	`df.tail(5)`
Afficher des indicateurs statistiques pour df.	`df.describe()`
Afficher des indicateurs statistiques pour la variable (colonne) nommée C.	`df['C'].describe()`
Calculer la moyenne de la variable C.	`df['C'].mean()`
Calculer la médiane de la variable C.	`df['C'].median()`
Calculer la somme de la variable C.	`df['C'].sum()`
Compter le nombre de valeurs de la variable quantitative C.	`df['C'].count()`
Compter le nombre de valeurs de la variable qualitative C.	`df['C'].value_counts()`
Trier le dataframe df par ordre croissant selon la variable C.	`df.sort_values(by = 'C ')`
Trier le dataframe df par ordre décroissant selon la variable C.	`df.sort_values(by = 'C ', ascending = False)`
Extraction d'un sous-tableau sous condition – Tableau croisé	

Créer un dataframe df2 correspondant aux lignes de df où la variable C est non nulle (trois écritures possibles).	`df2 = df.query('C =! 0')`
	`df2 = df[df.C =! 0]`
	`df2 = df[df['C'] =! 0]`

Créer un dataframe df2 correspondant aux lignes de df où la variable A est supérieure à 2 et la variable B inférieure à 6 (pour « ou » utiliser |).	`df2 = df[(df.A > 2) & (df.B < 6)]`
Créer un dataframe df2 échantillon aléatoire sans remise de taille 100 de df.	`import random` `df2 = df.sample(n = 100)`
Regroupement des données selon les valeurs de la variable C (tableau croisé).	`df.groupby('C')`

Représentations graphiques	
Importer le module matplotlib.	`import matplotlib.pyplot as plt`
Créer un nuage de points de la variable C en fonction de la variable A.	`df.plot.scatter(x = 'A', y = 'C')`
Créer un histogramme représentant les valeurs de la variable quantitative C réparties en 20 classes égales.	`df.hist(column = 'C', figsize = (9, 6), bins = 20)`
Créer un diagramme en barres représentant les effectifs des différentes valeurs de la variable qualitative C.	`df['C'].value_counts().plot.bar()`
Créer un diagramme circulaire représentant les parts relatives des différentes valeurs de la variable C.	`df['C'].value_counts().plot.pie()`
Créer un diagramme en boîte des valeurs de la variable C.	`df.boxplot(column = 'C ')`
Installer le module Folium permettant de situer des données géographiques sur une carte dynamique. Saisir dans la console :	`pip install folium`
Importer le module Folium et sa bibliothèque Plugins.	`import folium` `from folium import plugins`
Créer une carte centrée sur le point de latitude 48.8 et de longitude 2.3 selon le modèle « cartodbpositron » avec zoom initial de 12..	`carte = folium.Map(location = [48.8, 2.3], tiles = 'cartodbpositron', zoom_start = 12)`
Ajouter à la carte interactive les points de la liste de coordonnées géographiques Lieux selon un modèle « carte de chaleur ».	`plugins.HeatMap(Lieux, radius = 10, blur = 0).add_to(carte)`
Enregistrer la carte selon le chemin précisé dans … pour l'observer en ligne.	`carte.save('C:/…/carte.html')`

Pour davantage de détails sur les fonctions pandas, on peut consulter la documentation en ligne : https://pandas.pydata.org/pandas-docs/stable/api.html

11. Entreprises et salaires en France

Les données proviennent du site kaggle.com[3] et sont issues de l'INSEE : nombre d'établissements par secteur d'activité et par taille en 2014 (champ des activités marchandes hors agriculture).
https://www.insee.fr/fr/statistiques/1893274

Les objectifs sont :
– représenter et déterminer les villes ou les régions les plus porteuses d'emplois ;
– étudier les salaires selon le lieu, le genre (homme/femme) et le type de métier.

Trois fichiers csv sont lus dans les dataframe « entreprises », « geographie » et « salaires ».

```
1  import matplotlib.pyplot as plt
2  import pandas
3
4  # Lecture des donnees
5  entreprises = pandas.read_csv('entreprises.csv', sep = ',',
   encoding = 'latin-1')
6  geographie = pandas.read_csv('geographie.csv', sep = ',',
   encoding = 'latin-1')
7  salaires = pandas.read_csv('salaires.csv', sep = ',',
   encoding = 'latin-1')
```

Description des trois fichiers de données

Fichier entreprises.csv

Ce fichier fournit des informations sur le nombre d'entreprises (hors agriculture) dans chaque commune française.
Les variables sont :
CODGEO : code géographique pour la ville, correspond à la variable code_insee du fichier geographie.csv ;
LIBGEO : nom de la ville ;
REG : numéro de la région ;
DEP : numéro du département ;
E14TST : nombre total d'entreprises dans la ville ;
E14TS0ND : nombre d'entreprises de taille inconnue ou nulle dans la ville ;
E14TS1 : nombre d'entreprises avec 1 à 5 employés dans la ville ;
E14TS6 : nombre d'entreprises avec 6 à 9 employés dans la ville ;
E14TS10 : nombre d'entreprises avec 10 à 19 employés dans la ville ;
E14TS20 : nombre d'entreprises avec 20 à 49 employés dans la ville ;
E14TS50 : nombre d'entreprises avec 50 à 99 employés dans la ville ;
E14TS100 : nombre d'entreprises avec 100 à 199 employés dans la ville ;
E14TS200 : nombre d'entreprises avec 200 à 499 employés dans la ville ;
E14TS500 : nombre d'entreprises avec plus de 500 employés dans la ville.

Fichier geographie.csv

Ce fichier fournit des informations géographiques sur les communes françaises, notamment la latitude et la longitude.
Les variables sont :

[3] Site d'une start-up californienne organisant des compétitions en science des données.

EU_circo : nom de la circonscription de l'Union Européenne ;
code_région : code de la région de rattachement de la ville ;
nom_région : nom de la région de rattachement de la ville ;
chef.lieu_région : nom du chef-lieu de région ;
numéro_département : numéro du département de rattachement de la ville ;
nom_département : nom du département de rattachement de la ville ;
préfecture : nom de la préfecture du département ;
numéro_circonscription : numéro de la circonscription ;
nom_commune : nom de la ville (ou commune) ;
codes_postaux : codes postaux de la ville ;
code_insee : code INSEE de la ville ;
latitude : latitude GPS de la ville ;
longitude : longitude GPS de la ville ;
éloignement : coefficient d'éloignement (à préciser si nécessaire).

Fichier salaires.csv

Ce fichier fournit les salaires nets (2014) par commune, catégorie d'emploi, âge et genre.
Les variables sont :
CODGEO : code géographique pour la ville, correspond à la variable code_insee du fichier geographie.csv ;
LIBGEO : nom de la ville ;
SNHM14 : salaire net horaire moyen ;
SNHMC14 : salaire net horaire moyen des cadres supérieurs ;
SNHMP14 : salaire net horaire moyen des professions intermédiaires ;
SNHME14 : salaire net horaire moyen des employés ;
SNHMO14 : salaire net horaire moyen des ouvriers ;
SNHMF14 : salaire net horaire moyen des femmes ;
SNHMFC14 : salaire net horaire moyen des femmes cadres supérieures ;
SNHMFP14 : salaire net horaire moyen des femmes de profession intermédiaire ;
SNHMFE14 : salaire net horaire moyen des femmes employées ;
SNHMFO14 : salaire net horaire moyen des femmes ouvrières ;
SNHMH14 : salaire net horaire moyen des hommes ;
SNHMHC14 : mean net salary per hour for masculin executive
SNHMHP14 : mean net salary per hour for masculin middle manager
SNHMHE14 : mean net salary per hour for masculin employee
SNHMHO14 : mean net salary per hour for masculin worker
SNHM1814 : salaire net horaire moyen pour les 18-25 ans ;
SNHM2614 : salaire net horaire moyen pour les 26-50 ans ;
SNHM5014 : salaire net horaire moyen pour les plus de 50 ans ;
SNHMF1814 : salaire net horaire moyen pour les femmes entre 18 et 25 ans ;
SNHMF2614 : salaire net horaire moyen pour les femmes entre 26 et 50 ans ;
SNHMF5014 : salaire net horaire moyen pour les femmes de plus de 50 ans ;
SNHMH1814 : salaire net horaire moyen pour les hommes entre 18 et 25 ans ;
SNHMH2614 : salaire net horaire moyen pour les hommes entre 26 et 50 ans ;
SNHMH5014 : salaire net horaire moyen pour les hommes de plus de 50 ans.

1. Pour chacun des trois fichiers, examiner les données (taille, type des variables, 5 premières lignes ou de façon aléatoire).

2. Préparation du dataframe « entreprises »
a. Certaines lignes de la variable CODGEO sont incorrectes et contiennent des caractères non numériques. Entrer l'instruction suivante permettant de ne sélectionner dans le dataframe « entreprises » que les lignes dont la variable CODGEO peut être transformée en nombre.

```
entreprises = entreprises[entreprises['CODGEO'].apply(lambda x:
str(x).isdigit())]
```

La variable CODGEO est de type « object » (chaîne de caractères). Changer le type de cette variable en « entier » (int).
b. Les critères pour définir la taille d'une entreprise peuvent varier d'un pays à l'autre. La Commission européenne définit les catégories : micro (moins de 10 employés), petite (moins de 50 employés), médium (moins de 250 employés), grande (moins de 1 000 employés) et très grande (plus de 1 000 employés).
Créer quatre nouvelles colonnes nommées Micro, Petite, Medium et Grande correspondant au mieux aux critères de la Commission européenne.
c. Créer quatre colonnes Micro%, Petite%, Medium% et Grande% fournissant le pourcentage du nombre total d'entreprises correspondant à chaque catégorie.
d. Restreindre le dataframe « entreprises » aux variables 'CODGEO', 'LIBGEO', 'REG', 'DEP', 'Somme', 'Micro', 'Petite', 'Medium', 'Grande', 'Micro%', 'Petite%', 'Medium%' et 'Grande%'.

3. Préparation du dataframe « geographie »
Supprimer les variables inutiles à notre étude en exécutant l'instruction suivante.

```
geographie.drop(['EU_circo', 'code_region', 'eloignement',
'numero_departement', 'nom_departement', 'prefecture',
'numero_circonscription', 'codes_postaux'], axis=1, inplace=True)
```

Indiquer à quoi correspondent les instructions suivantes, puis les exécuter.

```
geographie['longitude'] = geographie['longitude'].apply(lambda x:
str(x).replace(',','.'))
supprimer = geographie['longitude'] == '-'
geographie.drop(geographie[supprimer].index, inplace=True)
geographie.dropna(subset = ['longitude', 'latitude'], inplace=True)
geographie['longitude'] = geographie['longitude'].astype(float)
```

Enfin, supprimer les doublons ayant le même code INSEE.

```
geographie.drop_duplicates(subset=["code_insee"], keep="first", inplace=True)
```

4. Préparation du dataframe « salaires »
De même que pour le dataframe « entreprises », convertir la variable CODGEO au type entier.

5. Fusion des dataframe « entreprises » et « geographie »
La fonction « merge » permet de fusionner deux dataframes. Exécuter l'instruction suivante puis observer le dataframe « data » obtenu.

```
data = entreprises.merge(geographie, how='left', left_on = 'CODGEO',
right_on='code_insee')
```

6. Répartition géographique des entreprises française

a. On souhaite réaliser une carte dynamique des 1 000 communes les plus porteuses d'emploi en France.

Exécuter les instructions suivantes, en remplaçant, à la ligne 52, le chemin par celui qui convient pour l'affichage de la carte (cet affichage nécessite une connexion Internet).

```
38 #Carte dynamique des 1000 communes les plus porteuses d'emploi
39 data.dropna(subset = ['longitude', 'latitude', 'Somme'],
   inplace=True)
40 import folium
41 lat_moy = data['latitude'].mean()
42 long_moy = data['longitude'].mean()
43 carte = folium.Map(location = [lat_moy, long_moy], zoom_start = 6,
   tiles = 'Stamen Terrain')
44 top1000 = data.sort_values(by = 'Somme').tail(1000)
45
46 for k in top1000.index.values :
47     latitude = top1000.loc[k, 'latitude']
48     longitude = top1000.loc[k, 'longitude']
49     rayon = top1000.loc[k, 'Somme']/1000
50     folium.CircleMarker([latitude, longitude], radius =
   rayon).add_to(carte)
51
52 carte.save('C:/Users/dutar/Desktop/carte.html')
```

b. Modifier le programme précédent de façon que l'aire des disques soit proportionnelle au nombre d'entreprises (et non le rayon comme précédemment).

7. Classement des régions les plus porteuses d'emploi

Déterminer les régions les plus porteuses d'emplois en termes de nombre d'entreprises. Comparer la moyenne et la médiane du nombre d'entreprises par région. Interpréter.

8. Ecarts de salaire selon le lieu et le genre

a. Rechercher les 10 communes avec les plus hauts et les plus bas salaires moyens.

b. Etudier les écarts de salaire entre les hommes et les femmes.

12. Paradise Papers

Les données des « Paradise Papers », publiées par le Consortium international des journalistes d'investigation en novembre 2017 concernent des investissements « offshore »[4]. On peut obtenir sur le site kaggle.com[5] des fichiers csv correspondant à ces données.
Nous explorons avec Python quatre fichiers nommés entity, officer, address et edges. Le fichier geo_papers fournit des données géographiques.

```
1 import matplotlib.pyplot as plt
2 import pandas
3
4 # Lecture des donnees
5 entites = pandas.read_csv('entity.csv', sep = ',')
6 executeurs = pandas.read_csv('officer.csv', sep = ',')
7 adresses = pandas.read_csv('address.csv', sep = ',')
8 liens = pandas.read_csv('edges.csv', sep = ',')
9 geographie = pandas.read_csv('geo_papers.csv', sep = ',', decimal =
',')
```

[4] On peut à ce propos consulter les article suivants du journal Le Monde : 05/11/2017 *Les « Paradise Papers » : nouvelles révélations sur les milliards cachés de l'évasion fiscale* ou 14/02/2018 *« Paradise Papers » : des dizaines de milliers de sociétés offshore rendues publiques dans la « Offshore Leaks Data Base »*.
[5] Site d'une start-up californienne organisant des compétitions en science des données.

1. Etude des pays impliqués dans les « Paradise Papers »

a. Le fichier entity.csv correspond aux compagnies offshore jouant le rôle d'écran dans un paradis fiscal.
Déterminer la taille et étudier les variables de ce fichier.
En utilisant le tableau « paradis » suivant, déterminer les principaux paradis fiscaux impliqués dans les « Paradise Papers » et effectuer une représentation graphique.

```
paradis = entites['n.countries'].value_counts()
```

b. Le fichier officer.csv correspond aux exécuteurs, c'est-à-dire aux entreprises ou particuliers donneurs d'ordre pour des « clients » souhaitant échapper au fisc de leur pays.
Déterminer la taille et étudier les variables de ce fichier.
Déterminer les principaux pays exécuteurs impliqués dans les « Paradise Papers » et effectuer une représentation graphique.

c. Le fichier address.csv donne la localisation des « clients », c'est-à-dire des commanditaires, ceux à qui profite la fraude.
Déterminer la taille et étudier les variables de ce fichier.
Déterminer les principaux pays où sont localisés les « clients » impliqués dans les « Paradise Papers » et effectuer une représentation graphique.

2. Représentation sur une carte des principaux pays impliqués

a. Le dataframe geographie contient les coordonnées géographiques des 22 principaux pays cités.
Etudier la dimension, les variables et leur type de ce tableau.

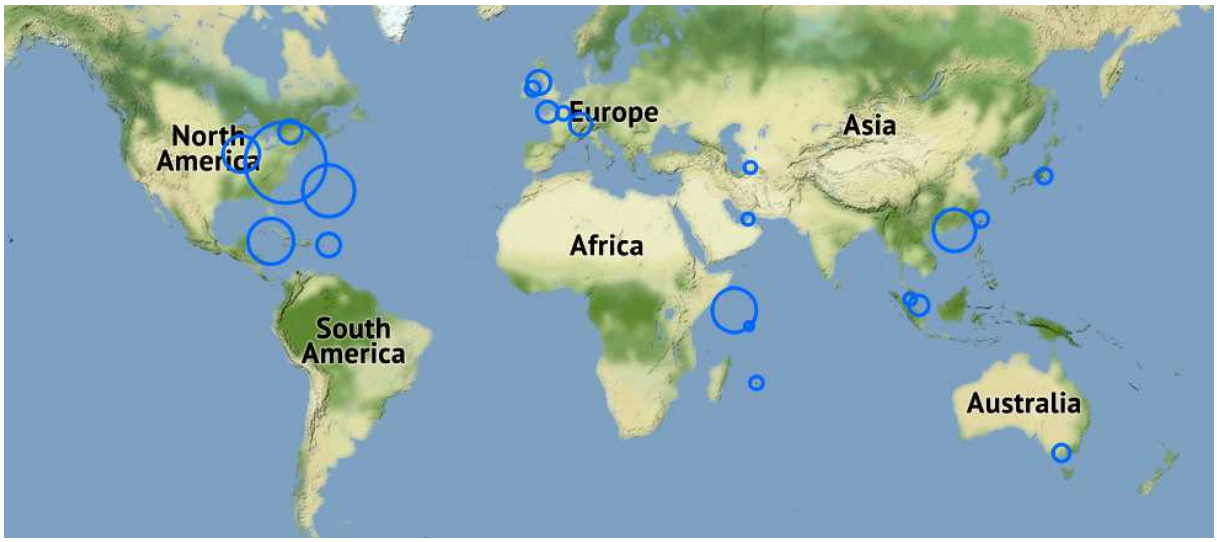
b. On souhaite représenter ces pays sur une carte avec un disque d'aire proportionnelle au nombre de fois où le pays est cité comme intervenant dans les « Paradise Papers ».
Pour cela, on commence par ajouter les quatre colonnes suivantes au dataframe « geographie ».

```
18 # 1. Ajout de nouvelles colonnes au dataframe geographie
19 col1 = []
20 for pays in geographie['state'] :
21     if pays in paradis.index.values :
22         col1.append(paradis[pays])
23     else :
24         col1.append(0)
25 col2 = []
26 for pays in geographie['state'] :
27     if pays in pays_executeurs.index.values :
28         col2.append(pays_executeurs[pays])
29     else :
30         col2.append(0)
31 col3 = []
32 for pays in geographie['state'] :
33     if pays in pays_clients.index.values :
34         col3.append(pays_clients[pays])
35     else :
36         col3.append(0)
37 geographie['nb paradis'] = col1
38 geographie['nb executeurs'] = col2
39 geographie['nb clients'] = col3
40 col4 = geographie.apply(lambda row : row['nb paradis'] + row['nb
   executeurs'] + row['nb clients'], axis = 1)
41 geographie['poids'] = col4
```

A quoi correspondent chacune des quatre nouvelles colonnes ?

c. Le programme suivant produit une carte avec des disques d'aires proportionnelles au nombre de fois où chaque pays est cité comme intervenant dans les « Paradise Papers » (veiller à indiquer le chemin correct à la ligne 54 et à être connecté à Internet pour visualiser la carte).

```
43 # 2. Carte des paradis fiscaux
44 import folium
45 carte_paradis = folium.Map(location = [0, 0], zoom_start = 2, tiles
   = 'Stamen Terrain')
46
47 from math import sqrt, pi
48 for k in geographie.index.values :
49     latitude = geographie.loc[k, 'latitude']
50     longitude = geographie.loc[k, 'longitude']
51     rayon = sqrt(geographie.loc[k, 'poids']/(10 * pi))
52     folium.CircleMarker([latitude, longitude], radius =
   rayon).add_to(carte_paradis)
53
54 carte_paradis.save('C:/Users/dutar/Desktop/carte_paradis.html')
```



Justifier le calcul effectué à la ligne 51.

3. Représentation des liens sur une carte
Le fichier edges.csv fournit les liens existant entre les différents acteurs. Il comporte 364 456 lignes, chacune indiquant un lien entre le « nœud 1 » (un client, un exécuteur, une entité) et le « nœud 2 » du réseau.

a. Examiner les premières lignes du dataframe liens.

b. Le programme suivant examine les 10 000 premières lignes du dataframe liens, pour représenter sur une carte les liens existants lorsque les pays concernés figurent dans la table geographie (10 000 a été choisi pour des raisons de temps de calcul, le graphique obtenu donnant une assez bonne idée des principaux liens existant dans les Paradise Papers).
Compléter les lignes 85 et 87 du programme, puis l'exécuter.

```
# 3. Representation des liens sur une carte
carte_liens = folium.Map(location=[0,0],zoom_start = 2,tiles='Stamen Terrain')

for k in range(10000):
    # On cherche dans les 3 fichiers executeurs, adresses
    # et entites, le pays correspondant au noeud 1
    # reset_index(drop = True) permet de reinitialiser l'index
    noeud1 = liens.loc[k,'node_1']
    try :
        if executeurs[executeurs['n.node_id'] == noeud1]['n.countries'].empty == False:
            pays1 = executeurs[executeurs['n.node_id'] == noeud1]['n.countries'].reset_index(drop = True)
        elif adresses[adresses['n.node_id'] == noeud1]['n.countries'].empty == False:
            pays1 = adresses[adresses['n.node_id'] == noeud1]['n.countries'].reset_index(drop = True)
        elif entites[entites['n.node_id'] == noeud1]['n.countries'].empty == False:
            pays1 = entites[entites['n.node_id'] == noeud1]['n.countries'].reset_index(drop = True)
        pays1 = pays1[0]
    except:
        pays1 = ''
    # si le pays 1 est dans le fichier geographie, on cherche
    # dans les 3 fichiers le pays correspondant au noeud 2
    if geographie[geographie['state']== pays1].empty == False:
        noeud2 = liens.loc[k,'node_2']
        try:
            if executeurs[executeurs['n.node_id'] == noeud2]['n.countries'].empty == False:
                pays2 = executeurs[executeurs['n.node_id'] == noeud2]['n.countries'].reset_index(drop = True)
            elif adresses[adresses['n.node_id'] == noeud2]['n.countries'].empty == False:
                pays2 = adresses[adresses['n.node_id'] == noeud2]['n.countries'].reset_index(drop = True)
            elif entites[entites['n.node_id'] == noeud2]['n.countries'].empty == False:
                pays2 = entites[entites['n.node_id'] == noeud2]['n.countries'].reset_index(drop = True)
            pays2 = ...
        except:
            pays2 = ...
```

```
88      # Si les pays 1 et 2 sont dans le fichier geographie, on
89      # recupere leur latitude et longitude et on les relie
90      if geographie[geographie['state']== pays1].empty == False:
91          if geographie[geographie['state']== pays2].empty == False:
92              pays = geographie[geographie['state']==
    pays1].reset_index(drop = True)
93              lat1 = pays['latitude'][0]
94              long1 = pays['longitude'][0]
95              pays = geographie[geographie['state']==
    pays2].reset_index(drop = True)
96              lat2 = pays['latitude'][0]
97              long2 = pays['longitude'][0]
98              folium.PolyLine([[lat1,long1],[lat2,long2]],
    color='red').add_to(carte_liens)
99
100 carte_liens.save('C:/Users/dutar/Desktop/carte_liens.html')
```

Conclusion

Les programmes de mathématiques font une part croissante à l'éducation du citoyen, notamment dans le cadre du socle commun. Les exemples développés montrent la place de premier ordre qu'occupe dans ce cadre l'enseignement de la statistique, des probabilités, de l'algorithmique et de la programmation, enseignement qui, lui même, a pris de l'importance dans les programmes de mathématiques. Le développement des « big data », du rôle de l'algorithmique et de l'intelligence artificielle dans notre quotidien offre de nouvelles perspectives d'apprentissage pour « armer » le futur citoyen des connaissances nécessaires à son jugement critique et pour qu'il puisse jouer un rôle actif et éclairé dans la société.

Éléments de réponse et exploitation pédagogique

1. Éducation civique et aire du disque

Critique « civique » pouvant être développée sur cet exemple. Ici pas de difficulté sur l'origine des données (officielle) ni de tricherie sur leurs valeurs (reproduites). Mais la représentation graphique du journal a pour effet de polariser l'attention sur le seul débat entre les deux blocs dominants. Il y a donc eu une représentation « faussement scientifique » ; le citoyen doit se méfier des messages subliminaires véhiculés par le graphisme et revenir aux données.

C'est aussi une occasion d'exercice élémentaire sur la géométrie du cercle, sur les rapports et sur la linéarité. Pour l'un et l'autre des « dessins » (celui du Monde et le correct), on peut :

– faire mesurer les diamètres des disques et calculer leurs aires (par exemple en centimètres carrés) ;

– prendre des couples de familles politiques et comparer les rapports des aires avec ceux des pourcentages de suffrages correspondants.

On peut aussi faire représenter dans le plan les points associés à chaque famille politique, d'abscisse le pourcentage de suffrages et d'ordonnée l'aire du disque.

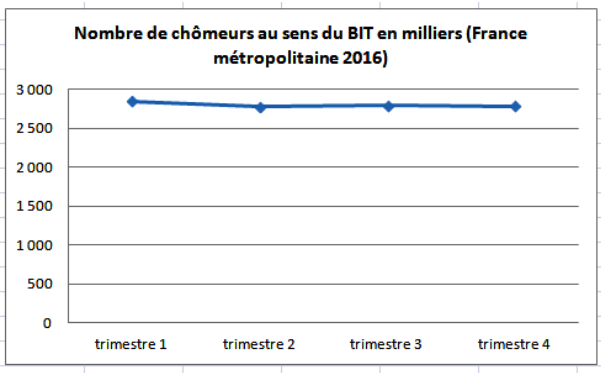
2. Diagrammes trompeurs

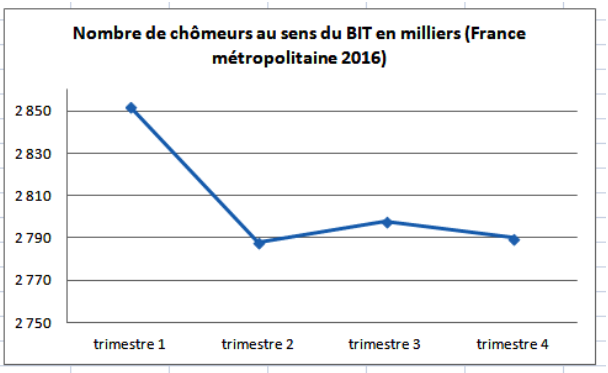
1. Lorsqu'on double les longueurs, on multiplie par huit le volume.

2. L'axe des ordonnées, non gradué, ne commence visiblement pas à zéro. Ceci peut induire un effet trompeur sur les variations relatives observées.

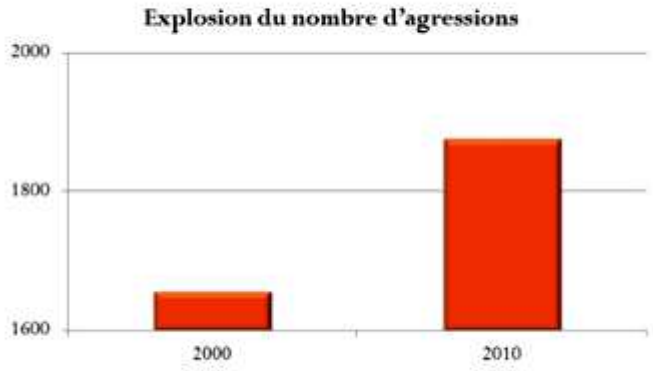
3. Chômage

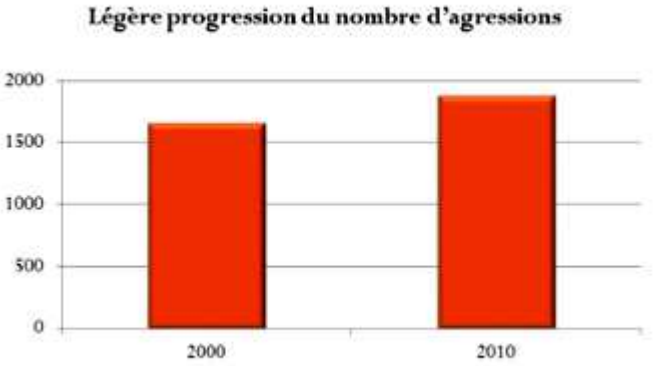
Exemples de graphiques :

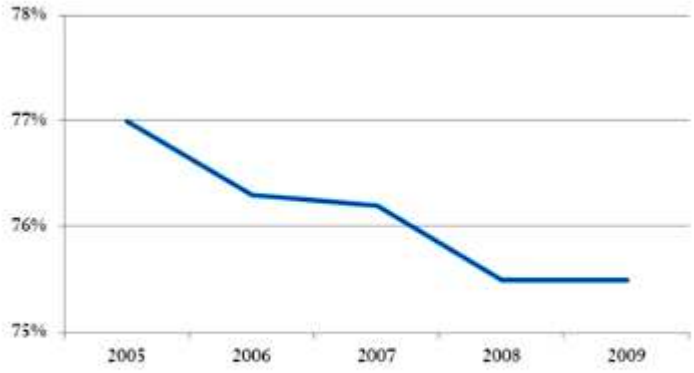




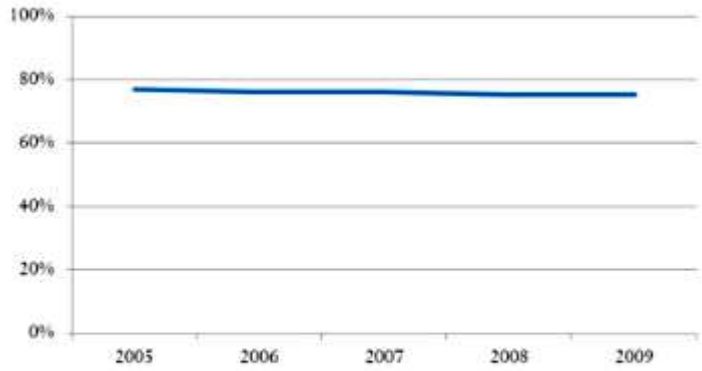
4. Autres exemples



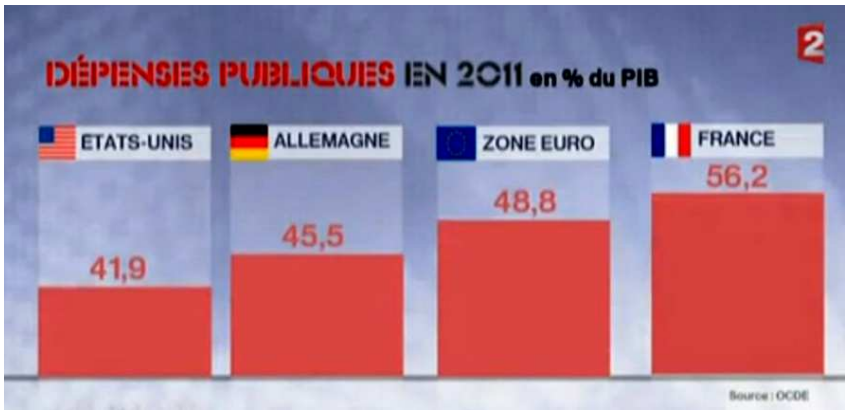


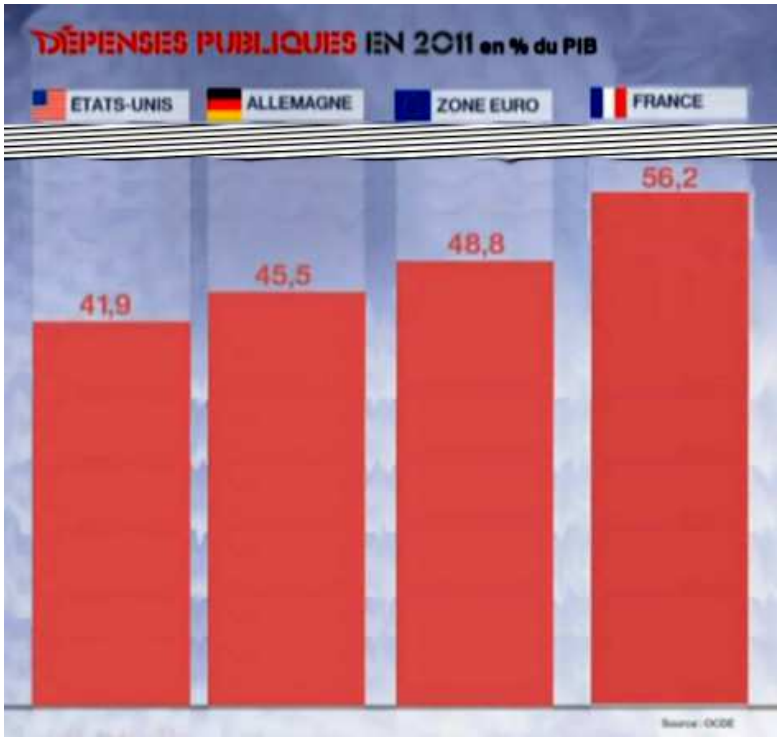


Niveau de prise en charge sécurité sociale.
Les Cahiers MGEN, 269, décembre 2010, p. 18



Niveau de prise en charge sécurité sociale.
Les Cahiers MGEN, 269, décembre 2010, p. 18





5. Abaque de calcul de la marge d'erreur et simulations

1. Fonction echantillon

```
3  def echantillon(n,p):
4      s = 0
5      for k in range(n):
6          if random.random() < p:
7              s = s + 1
8      return s / n
```

2. Exemple de fréquence *f* dans le cas *n*= 1000 et *p*=0,2.

```
>>> echantillon(1000,0.2)
0.189
```

Dans le tableau, on peut lire que la marge est égale à 0,025.
La fourchette de sondage est donc l'intervalle [0.164 ; 0,214]. Elle contient *p*.
3. Fonction test_fourchette :

```
10 def test_fourchette(n,p,marge):
11     f = echantillon(n,p)
12     if p>= f - marge and p<= f + marge:
13         return 1
14     else:
15         return 0
```

4. Sur les premières valeurs testées, la fourchette de sondage semble contenir *p*.

```
>>> test_fourchette(1000,0.2,0.025)
1

>>> test_fourchette(1000,0.2,0.025)
1

>>> test_fourchette(1000,0.2,0.025)
1

>>> test_fourchette(1000,0.2,0.025)
1

>>> test_fourchette(1000,0.2,0.025)
1
```

5. La fonction mille_fourchettes simule 1000 fourchettes de sondages et renvoie la proportion de celles qui contiennent *p*. On remarque que cette proportion est proche de 0,95.

```
>>> mille_fourchettes(1000,0.2,0.025)
0.947

>>> mille_fourchettes(1000,0.2,0.025)
0.952

>>> mille_fourchettes(1000,0.4,0.03)
0.95

>>> mille_fourchettes(500,0.5,0.045)
0.952
```

6. Contrairement à ce que dit la phrase qui suit le tableau, on ne peut pas affirmer que la proportion p est contenue dans la fourchette de sondage. Il semble que la probabilité d'obtenir une fourchette contenant p soit proche de 0,95 ; d'où le titre du tableau.
7. Le « 19 fois sur 20 » n'est autre que le « 95% de chance » du tableau de l'Ifop.

6. Sourcier or not sourcier ?

Énoncé niveau seconde

1. On a $p = \dfrac{1}{5} = 0{,}2$.

2. a) On a simulé 200 échantillons de taille 30.

b) De nombreux points sont situés au-dessus de 0,25. Obtenir au moins 25 % de réponses exactes n'est pas rare.

c) Obtenir, en répondant au hasard, au moins 40 % de réponses exactes est possible mais n'a pas été observé sur les 200 simulations. Cet événement est rare.

d) L'intervalle $\left[p - \dfrac{1}{\sqrt{n}},\ p + \dfrac{1}{\sqrt{n}}\right]$ correspond environ à [0,018 ; 0,383]. En répondant au hasard, on a, dans environ 95 % des cas, entre 1,8 % et 38,3 % de bonnes réponses.

2. La fréquence de bonnes réponses observées est $\dfrac{8}{30} \approx 0{,}267$. Cette fréquence est comprise dans l'intervalle de fluctuation à 95 % lorsque l'on répond au hasard. Le score du pseudo sorcier est donc insuffisant pour considérer qu'il ne répond pas au hasard.

Énoncé niveau première

2. X suit la loi binomiale de paramètres $n = 30$ et $p = 0{,}2$.

3. Un tableur, par exemple, permet de tabuler la loi binomiale comme ci-après.

B10 : =LOI.BINOMIALE(A10;30;0,2;VRAI)

	A	B	C	D
1	k	P(X <= k)	P(X > k)	
2	0	0,00123794	0,99876206	
3	1	0,01052249	0,98947751	
4	2	0,04417899	0,95582101	
5	3	0,12271081	0,87728919	
6	4	0,25523325	0,74476675	
7	5	0,42751244	0,57248756	
8	6	0,60696992	0,39303008	
9	7	0,76079062	0,23920938	
10	8	0,87134925	0,12865075	
11	9	0,93891285	0,06108715	
12	10	0,97438374	0,02561626	
13	11	0,99050688	0,00949312	
14	12	0,99688895	0,00311105	
15	13	0,99909813	0,00090187	

Plusieurs raisonnements sont possibles.

• On constate que $P(X \geq 8) \approx 0{,}24$. Ainsi, une personne répondant au hasard à 24 % de chances de faire un score supérieur ou égal à celui du pseudo sourcier. On peut considérer cette probabilité comme trop importante pour attribuer au soucier un pouvoir particulier.

• On peut chercher à mettre en place une règle de décision en amont de l'expérience réalisée. On formule l'hypothèse que la personne répond au hasard. Sous cette hypothèse, la probabilité que le nombre de bonnes réponses soit dans l'intervalle [0, 9] est environ 0,94. On peut adopter la règle de décision suivante :

– si le nombre de bonnes réponses est supérieur ou égal à 10, on rejette l'hypothèse (et on considère dans ce cas que la personne ne répond pas au hasard) ;

– si le nombre de bonnes réponses est inférieur ou égal à 9, on ne rejette pas l'hypothèse et on considère que la personne répond au hasard.

Autrement dit, on attend au moins 10 bonnes réponses pour considérer le pseudo don du sourcier comme significatif. Dans cette procédure le risque de considérer, à tort, le pseudo don du sourcier comme significatif est de 6 %. On désigne cette procédure par « test unilatéral à droite au seuil de 6 % ».

Le soucier échoue à ce test.

• On peut utiliser la notion d'intervalle de fluctuation à 95 % au programme de la classe de première. Sous l'hypothèse d'une réponse au hasard, cet intervalle (bilatéral) correspond à la fluctuation des observations avec environ 2,5 % des résultats à gauche et à droite (sans dépasser les 2,5 %). L'intervalle de fluctuation à 95 % du nombre de résultats exacts sous l'hypothèse de réponses au hasard est : [2, 11].

On peut adopter la règle de décision suivante :

– si le nombre de bonnes réponses est supérieur ou égal à 12, on rejette l'hypothèse (et on considère dans ce cas que la personne ne répond pas au hasard) ;

– si le nombre de bonnes réponses est inférieur ou égal à 11, on ne rejette pas l'hypothèse et on considère que la personne répond au hasard.

Dans cette procédure le risque de considérer, à tort, le pseudo don du sourcier comme significatif est de 0,9 % (le caractère discret de la loi binomiale fait qu'on est très en-dessous des 2,5 % prévus à droite).

Le sourcier échoue à ce test, plus sévère que le précédent.

7. Le riz d'Emoto

1. $f = \frac{30}{50} = 0{,}6$.

2. Il faut comparer la fréquence f à la probabilité $p = 0{,}5$. Cela correspond à l'hypothèse selon laquelle les pensées positives non pas d'influence sur la conservation des aliments.

3.

$$p - \frac{1}{\sqrt{n}} = 0{,}5 - \frac{1}{\sqrt{50}} \approx 0{,}359$$

$$p + \frac{1}{\sqrt{n}} = 0{,}5 + \frac{1}{\sqrt{50}} \approx 0{,}641$$

La fréquence f est dans l'intervalle de fluctuation au seuil de 95%. L'expérience faite par Julien et Cécile ne fait pas apparaître de différence significative entre f et p.

4. Ce résultat va à l'encontre de la théorie d'Emoto.

8. Pseudo-sciences, sismologie et coïncidences

1. La proportion de jours sismiques en 2014, 2015 et 2016 est $p = \frac{134}{1096} \approx 0{,}122$.

2. L'intervalle de fluctuation au seuil de 95 % de la fréquence des jours sismiques sur un échantillon aléatoire de 129 dates est environ : [0,034 ; 0,21].

3. La fréquence de jours sismiques obtenue par l'astrologue est $f = \frac{19}{129} \approx 0{,}147$.

La fréquence f est comprise dans l'intervalle de fluctuation au seuil de 95 %. On peut considérer que l'astrologue n'a pas significativement fait mieux que le hasard.

4. Le programme suivant permet d'estimer que la probabilité de faire mieux que l'astrologue (au moins 20 succès) en répondant au hasard est environ 14 %.

```
1  import random
2
3  jours_sismiques = random.sample(range(1,1096),134)
   # liste fixee des 134 jours sysmiques parmi 1096
4
5  S = 0
6  for k in range(1000):
7      jours_hasard = random.sample(range(1,1096),129)
   # liste aleatoire et sans remise de 129 jours parmi
   1096
8      n = 0 # n compte le nombre d'elements de
   jours_hasard parmi les jours_sismiques
9      for j in jours_hasard:
10         if j in jours_sismiques:
11             n = n + 1
12     if n > 19:# on compare n aux 19 jours trouves
   par l'astrologue
13         S = S + 1
14 print(S /1000) # estimation de la probabilite de
   faire mieux que l'astrologue
```

Un exemple de résultat fourni par le programme :

```
0.142
```

Remarque pour le professeur :
Si X est la variable aléatoire correspondant au nombre de succès, $P(X > 19)$ vaut environ 0,156 si X suit la loi binomiale (tirages avec remise) $B(129 ; 0{,}122)$ et vaut environ 0,144 si X suit la loi hypergéométrique (tirages sans remise) $H(129 ; 0{,}122 ; 1096)$. Ces probabilités sont calculées à l'aide du tableur avec l'instruction :
=1-LOI.BINOMIALE(19;129;0,122;VRAI)
et en sommant les instructions :
=LOI.HYPERGEOMETRIQUE(x;129;134;1096) pour x allant de 0 à 19.
La probabilité 0,144 est bien estimée par le programme Python.

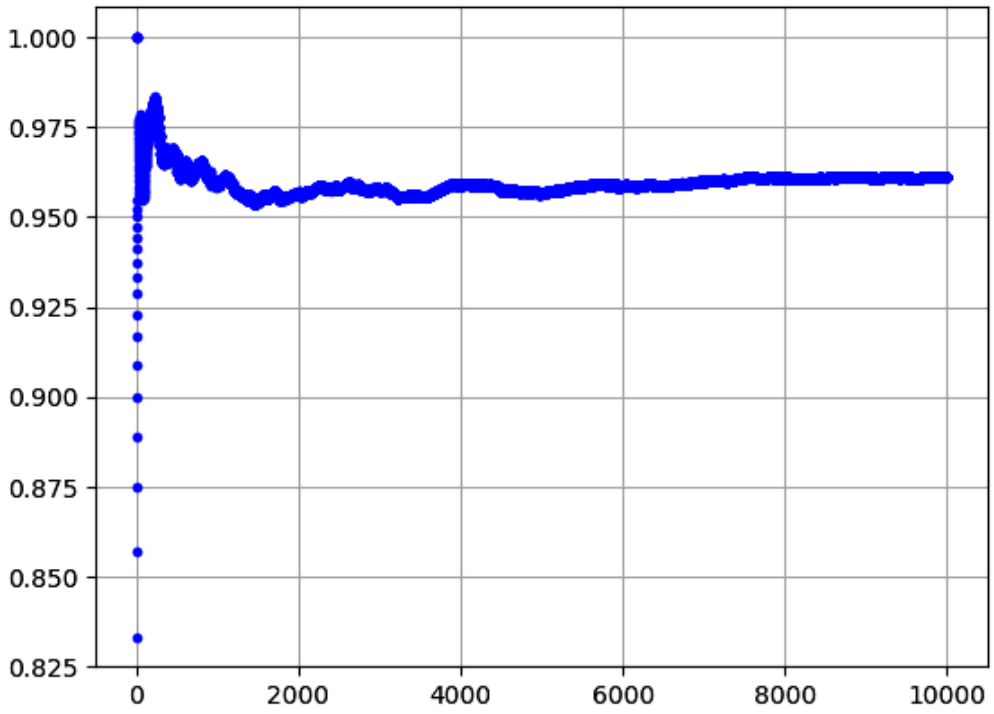
9. Psychogénéalogie et coïncidences

2. Exemples d'exécution de l'expérience :

```
>>> coincidence(10, 100)
[309, 304, 8, 82, 4, 180, 97, 157, 217, 60]
[302, 207, 216, 307, 322, 342, 85, 245, 315, 231, 98, 79, 110
, 349, 318, 21, 3, 240, 287, 226, 41, 25, 182, 169, 305, 185,
153, 212, 177, 144, 183, 148, 159, 102, 71, 155, 10, 74, 95,
101, 156, 152, 272, 108, 129, 301, 97, 286, 244, 336, 361, 35
2, 180, 267, 221, 277, 4, 350, 242, 354, 170, 117, 264, 323,
14, 8, 55, 265, 122, 70, 67, 288, 294, 248, 88, 33, 18, 198,
124, 171, 306, 316, 268, 186, 2, 320, 58, 76, 356, 247, 321,
201, 127, 42, 49, 131, 120, 218, 78, 69]
8
4
180
97
1
>>> coincidence(10, 100)
[104, 181, 30, 328, 141, 105, 314, 58, 82, 5]
[195, 227, 322, 104, 273, 126, 295, 264, 256, 116, 226, 311,
144, 121, 18, 225, 129, 146, 221, 117, 131, 33, 234, 128, 93,
60, 70, 44, 147, 2, 68, 166, 17, 162, 243, 85, 51, 208, 182,
114, 171, 292, 345, 27, 304, 76, 260, 347, 25, 15, 188, 72, 3
05, 45, 224, 209, 198, 119, 342, 290, 250, 7, 13, 363, 343, 1
70, 109, 215, 161, 105, 179, 356, 251, 312, 253, 315, 47, 53,
228, 219, 56, 193, 39, 202, 5, 338, 163, 142, 318, 31, 64, 23
3, 220, 143, 249, 248, 183, 245, 43, 200]
104
105
5
1
```

En renouvelant l'expérience ont constate que la coïncidence est extrêmement fréquente.

3. On peut produire le graphique suivant pour 10 000 répétitions de l'expérience. Cela confirme bien une estimation de la probabilité de l'événement E à 0,96, comme annoncé dans le texte.



4. Il suffit de répéter n fois avec $\frac{1}{\sqrt{n}} \leq 10^{-2}$ c'est-à-dire $n \geq 10\,000$.

10. Airbnb

Lecture du fichier csv et stockage des données dans le « dataframe » data.

```
1 import matplotlib.pyplot as plt
2 import pandas
3 # Lecture des donnees
4 data = pandas.read_csv('Airbnb_Paris.csv', sep = ';')
```

1. Nature des données

```
data.shape
data.dtypes
data.head(5)
```

```
>>> data.shape
(56450, 12)

>>> data.dtypes
id_location        int64
id_hote            int64
quartier          object
latitude         float64
longitude        float64
type              object
prix               int64
nuits_min          int64
nb_avis            int64
dernier_avis      object
avis_par_mois    float64
disponibilite      int64
dtype: object
```

Il y a 12 variables et 56 450 lignes.

Voici les 5 premières lignes.

```
>>> data.head(5)
   id_location   id_hote quartier   latitude  longitude                type
\
0      3508970  17667828  Reuilly  48.848748   2.376042  Maison/appt entier
1     13222966  12188988  Reuilly  48.847312   2.396370  Maison/appt entier
2      7337128   6403392  Reuilly  48.846610   2.407639  Maison/appt entier
3      5764597  14716027  Reuilly  48.846477   2.394853  Maison/appt entier
4      7861852  41437694  Reuilly  48.839819   2.406376         Piece privee

   prix  nuits_min  nb_avis dernier_avis  avis_par_mois  disponibilite
0    80          2       10   19/03/2017           0.46            282
1    55          1        3   29/07/2016           0.30              0
2   104          1       12   30/04/2016           0.58            365
3    45          3       13   31/03/2017           1.21              2
4    35          2        1   13/10/2015           0.06              0
```

On peut aussi examiner un échantillon aléatoire de 10 lignes.

```
>>> import random

>>> echantillon = data.sample(n = 10)

>>> echantillon.head(10)
       id_location   id_hote           quartier   latitude  longitude  \
27230     13813405  81101364             Elysee  48.879680   2.322678
39847     13430224  76584975       Observatoire  48.828370   2.332851
31582      7543672   7216863       Menilmontant  48.852682   2.408143
23665       427360   2124344             Louvre  48.864586   2.345891
26680      8529501  34988977             Elysee  48.871036   2.306869
50697      3478841  17505494  Buttes-Montmartre  48.890948   2.347938
29581      3371318  15319411             Temple  48.867593   2.353301
10026     12510181  23974056              Opera  48.882615   2.334681
42553      6351946  33092289         Popincourt  48.862372   2.386662
33913     14203059  51943712           Pantheon  48.851806   2.347510
                     type  prix  nuits_min  nb_avis dernier_avis  \
27230  Maison/appt entier    50          2         5   25/08/2016
39847        Piece privee    30          2         2   16/06/2016
31582        Piece privee    30          2        29   26/05/2016
23665  Maison/appt entier   130          4        73   06/03/2017
26680  Maison/appt entier    65          3         3   10/10/2016
50697  Maison/appt entier    80          1         0          NaN
29581  Maison/appt entier   195          4        23   24/10/2016
10026        Piece privee    50          2        33   22/03/2017
42553  Maison/appt entier    65          2         0          NaN
33913     Piece partagee    70          1         0          NaN

       avis_par_mois  disponibilite
27230           0.60              0
39847           0.20              0
31582           1.44              0
23665           1.22            101
26680           0.24              0
50697            NaN              0
29581           0.71            157
10026           2.86            315
42553            NaN              0
33913            NaN              0
```

On constate que certains avis ne sont pas renseignés ou que des locations sont à disponibilité nulle.

2. Données manquantes et valeurs aberrantes

```
    data.count()
>>> data.count()
id_location      56450
id_hote          56450
quartier         56442
latitude         56450
longitude        56450
type             56450
prix             56450
nuits_min        56450
nb_avis          56450
dernier_avis     42303
avis_par_mois    42301
disponibilite    56450
dtype: int64
```

Il manque 8 données de quartier seulement, mais 10 149 avis par mois.

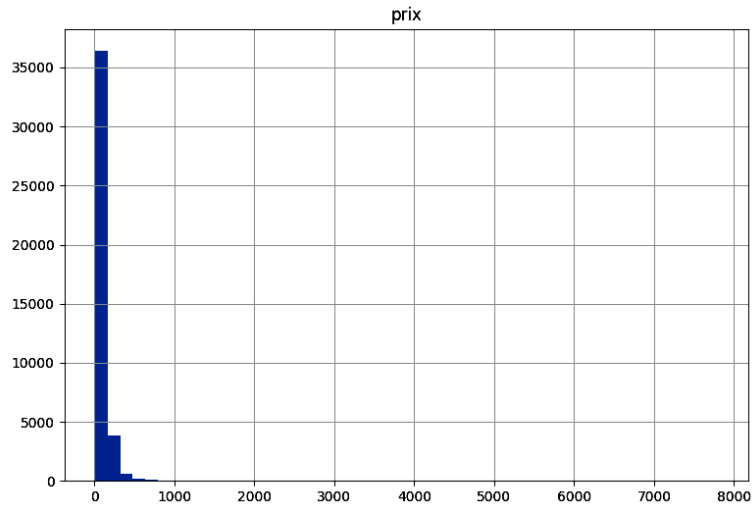
On supprime des données les locations à disponibilité nulle.

```
    data = data[data.disponibilite != 0]
>>> data = data[data.disponibilite != 0]

>>> data.count()
id_location      41220
id_hote          41220
quartier         41214
latitude         41220
longitude        41220
type             41220
prix             41220
nuits_min        41220
nb_avis          41220
dernier_avis     33357
avis_par_mois    33357
disponibilite    41220
dtype: int64
```

Il reste 41 220 locations disponibles.

D'éventuelles valeurs aberrantes peuvent être détectées en visualisant la répartition de certaines variables et en demandant leurs principaux indicateurs statistiques.

```
    data.hist(column = 'prix', figsize = (9,6), bins = 50)
    plt.show()
    data['prix'].describe()
```




```
>>> data['prix'].describe()
count    41220.000000
mean       102.874600
std        109.906818
min         10.000000
25%         58.000000
50%         80.000000
75%        117.000000
max       7790.000000
Name: prix, dtype: float64
```

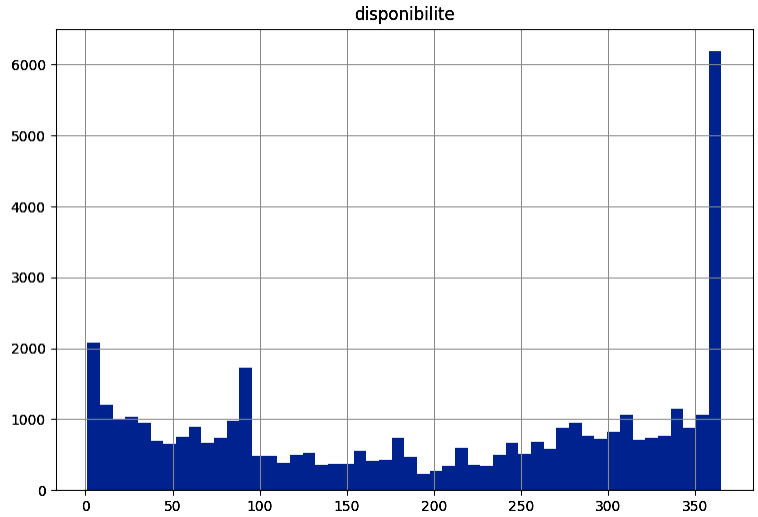
```
    prix = data.sort_values(by = 'prix', ascending=False)
    prix.head(5)
```

```
>>> prix = data.sort_values(by = 'prix', ascending=False)

>>> prix.head(5)
       id_location   id_hote         quartier   latitude  longitude
\
36052     13649901  48866007            Passy  48.868631   2.285734
47254     11640000  58922382           Bourse  48.866118   2.350599
38460     11268458   1112584     Observatoire  48.834013   2.324046
8681       9442661   2865311  Buttes-Chaumont  48.877121   2.386713
40636     14816222   4777697     Observatoire  48.830430   2.321907

                    type  prix  nuits_min  nb_avis dernier_avis  \
36052  Maison/appt entier  7790          7        0          NaN
47254        Piece privee  6070          2        0          NaN
38460        Piece privee  5000          1        0          NaN
8681   Maison/appt entier  5000        300       24   17/11/2016
40636  Maison/appt entier  3000          3        0          NaN

       avis_par_mois  disponibilite
36052            NaN            365
47254            NaN            365
38460            NaN             26
8681            1.41            178
40636            NaN            364
```

Les 5 appartements les plus chers ne sont peut-être pas des valeurs aberrantes.

```
    data.hist(column = 'disponibilite', figsize = (9,6), bins = 50)
```

```
plt.show()
data['disponibilite'].describe()
```

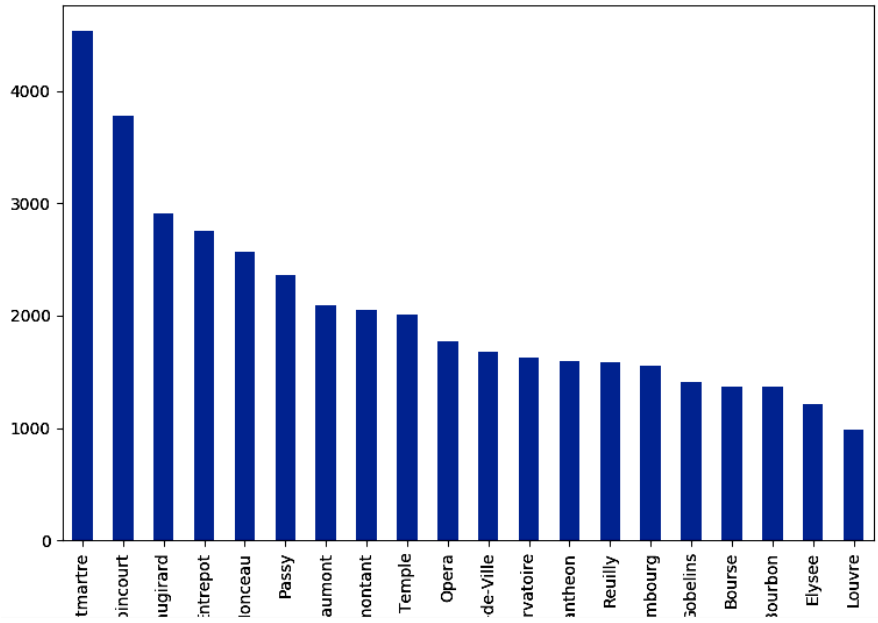



```
>>> data['disponibilite'].describe()
count    41220.000000
mean       200.002814
std        128.360714
min          1.000000
25%         78.000000
50%        216.000000
75%        326.000000
max        365.000000
Name: disponibilite, dtype: float64
```

3. et 4. Indicateurs statistiques, regroupements, graphiques, interprétations

Répartition par quartiers :

```
data['quartier'].value_counts().plot.bar()
```



Nombre d'avis par mois :

```
data['avis_par_mois'].describe()
```

```
>>> data['avis_par_mois'].describe()
count    33357.000000
mean         1.343011
std          1.457522
min          0.010000
25%          0.350000
50%          0.840000
75%          1.820000
max         18.000000
Name: avis_par_mois, dtype: float64
```

Avec un nombre moyen d'avis par mois égal à 1,34 et un prix moyen de 102,87 €, on peut estimer qu'une location Airbnb rapporte au minimum (en supposant que chaque locataire laisse un avis) en moyenne $1{,}34 \times 12 \times 102{,}87 \approx 2\,308$ € par an.

On peut affiner l'estimation en se limitant aux données pour lesquelles un avis au moins a été renseigné.

```
data2 = data[data.nb_avis > 0]
data2.describe()
```

```
>>> data2.describe()
       id_location       id_hote      latitude     longitude          prix
count  3.335800e+04  3.335800e+04  33358.000000  33358.000000  33358.000000
mean   8.404524e+06  2.717184e+07     48.863465      2.344537     99.196445
std    5.475130e+06  2.861701e+07      0.017790      0.032390     86.684861
min    2.525000e+03  2.626000e+03     48.811993      2.227926     10.000000
25%    3.370552e+06  6.028940e+06     48.851060      2.324800     55.000000
50%    7.900280e+06  1.598678e+07     48.864056      2.347904     79.000000
75%    1.353581e+07  3.856009e+07     48.877432      2.367272    110.000000
max    1.796882e+07  1.228353e+08     48.901698      2.467046   5000.000000

          nuits_min       nb_avis  avis_par_mois  disponibilite
count  33358.000000  33358.000000   33357.000000   33358.000000
mean       3.923407     23.316206       1.343011     198.019126
std       78.447253     35.745289       1.457522     126.022373
min        1.000000      1.000000       0.010000       1.000000
25%        2.000000      4.000000       0.350000      76.000000
50%        2.000000     10.000000       0.840000     216.000000
75%        3.000000     27.000000       1.820000     317.000000
max    10000.000000    488.000000      18.000000     365.000000
```

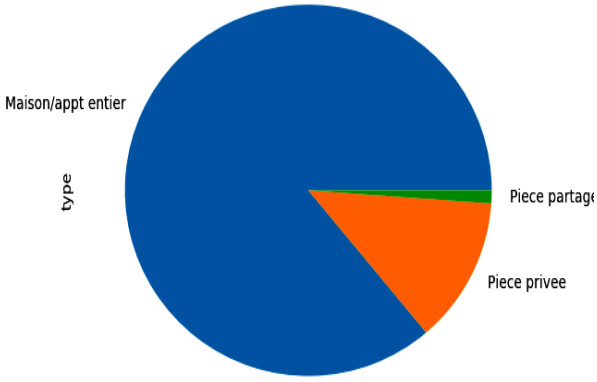
```
D = data2.describe()
D['nb_avis'] * D['prix']
```

```
>>> D['nb_avis'] * D['prix']
count    1.112756e+09
mean     2.312885e+03
std      3.098575e+03
min      1.000000e+01
25%      2.200000e+02
50%      7.900000e+02
75%      2.970000e+03
max      2.440000e+06
dtype: float64
```

Le gain moyen d'une location est 2 313 €. Le gain médian est 790 € par an et 75 % des locations rapportent moins de 2 970 € par an.

Type de location :

```
data['type'].value_counts().plot.pie()
```



L'immense majorité des locations sont des appartements entiers ou des pièces privées.

```
>>> type1 = data.query('type == "Maison/appt entier"')

>>> type1['type'].count()/data['type'].count()
0.86113537117903927

>>> type2 = data.query('type == "Piece privee"')

>>> type2['type'].count()/data['type'].count()
0.12755943716642407
```

Ces types de location représentent respectivement 86 % et 13 % des locations.

Locations très disponibles :

Une location est dite « très disponible » si elle l'est plus de 120 jours par an. Ces locations sont vraisemblablement très peu habitées.

```
>>> tresdispo = data.query('disponibilite > 120')

>>> tresdispo['disponibilite'].count() / data['disponibilite'].count()
0.63765162542455123
```

Environ 64 % des locations Airbnb à Paris sont « très disponibles ».

Hôtes multi-loueurs :

```
    data['id_hote'].value_counts()
>>> data['id_hote'].value_counts()
12984381     101
2288803       70
7612270       66
3972699       66
3943828       64
11593703      62
2667370       60
3971743       58
152242        54
7642792       53
288574        51
97916688      50
5044753       49
13013633      47
21630783      46
24495283      46
49869502      46
1322370       45
39922748      44
23025598      44
2107478       44
789620        42
67879895      41
```

Certains hôtes ont plus de 40 locations.

```
>>> hotes = data['id_hote'].value_counts()

>>> hotes.count()
33997

>>> hotes[hotes == 1].count()
31303

>>> hotes[hotes == 1].count() / hotes.count()
0.92075771391593375
```

92 % des hôtes ne proposent qu'une seule location, mais 8 % possèdent plusieurs locations.

Représentation sur une carte interactive avec le module folium :

Installation du module folium :
Entrer dans la console `pip install folium` .

Importation du module folium et de sa bibliothèque plugins :

```
import folium
from folium import plugins
```

Création d'une carte centrée sur le point moyen en latitude et longitude :

```
carte = folium.Map(location = [48.8638, 2.3448], tiles = 'cartodbpositron',
zoom_start = 12)
```

Création d'une liste des lieux de location Airbnb :

```
Lieux = []
for lat, long in zip(data['latitude'], data['longitude']) :
   Lieux.append((lat, long))
```

Figuration des lieux de location sur la carte selon le modèle d'une « carte de chaleur » :

```
plugins.HeatMap(Lieux, radius = 10, blur = 0).add_to(carte)
```

Sauvegarde de la carte pour lecture (connexion Internet nécessaire) :

```
carte.save('C:/Users/HP/Desktop/carte.html')
```

11. Entreprises et salaires en France

1.

```
>>> entreprises.shape
(36681, 14)

>>> entreprises.dtypes
CODGEO      object
LIBGEO      object
REG          int64
DEP         object
E14TST       int64
E14TS0ND     int64
E14TS1       int64
E14TS6       int64
E14TS10      int64
E14TS20      int64
E14TS50      int64
E14TS100     int64
E14TS200     int64
E14TS500     int64
dtype: object
```

```
>>> entreprises.head(5)
  CODGEO                     LIBGEO  REG DEP  E14TST  E14TS0ND  E14TS1  E14TS6  \
0  01001  L'Abergement-ClÃ©menciat   82  01      25        22       1       2
1  01002     L'Abergement-de-Varey   82  01      10         9       1       0
2  01004        AmbÃ©rieu-en-Bugey   82  01     996       577     272      63
3  01005      AmbÃ©rieux-en-Dombes   82  01      99        73      20       3
4  01006                  AmblÃ©on   82  01       4         4       0       0

   E14TS10  E14TS20  E14TS50  E14TS100  E14TS200  E14TS500
0        0        0        0         0         0         0
1        0        0        0         0         0         0
2       46       24        9         3         2         0
3        1        2        0         0         0         0
4        0        0        0         0         0         0
```

```
>>> geographie.shape
(36840, 14)

>>> geographie.dtypes
EU_circo                 object
code_region               int64
nom_region               object
chef.lieu_region         object
numero_departement       object
nom_departement          object
prefecture               object
numero_circonscription    int64
nom_commune              object
codes_postaux            object
code_insee                int64
latitude                float64
longitude                object
eloignement             float64
dtype: object

>>> geographie.head(5)
  EU_circo  code_region    nom_region chef.lieu_region numero_departement  \
0  Sud-Est           82  RhÃ´ne-Alpes             Lyon                   1
1  Sud-Est           82  RhÃ´ne-Alpes             Lyon                   1
2  Sud-Est           82  RhÃ´ne-Alpes             Lyon                   1
3  Sud-Est           82  RhÃ´ne-Alpes             Lyon                   1
4  Sud-Est           82  RhÃ´ne-Alpes             Lyon                   1

  nom_departement       prefecture  numero_circonscription  \
0             Ain  Bourg-en-Bresse                       1
1             Ain  Bourg-en-Bresse                       1
2             Ain  Bourg-en-Bresse                       1
3             Ain  Bourg-en-Bresse                       1
4             Ain  Bourg-en-Bresse                       1

           nom_commune codes_postaux  code_insee   latitude longitude  \
0             Attignat          1340        1024  46.283333  5.166667
1             Beaupont          1270        1029  46.400000  5.266667
2                BÃ©ny          1370        1038  46.333333  5.283333
3           BÃ©reyziat          1340        1040  46.366667      5.05
4  Bohas-Meyriat-Rignat          1250        1245  46.133333       5.4
```

```
>>> salaires.shape
(5136, 26)

>>> salaires.dtypes
CODGEO        object
LIBGEO        object
SNHM14       float64
SNHMC14      float64
SNHMP14      float64
SNHME14      float64
SNHMO14      float64
SNHMF14      float64
SNHMFC14     float64
SNHMFP14     float64
SNHMFE14     float64
SNHMFO14     float64
SNHMH14      float64
SNHMHC14     float64
SNHMHP14     float64
SNHMHE14     float64
SNHMHO14     float64
SNHM1814     float64
SNHM2614     float64
SNHM5014     float64
```

```
>>> salaires.head(5)
  CODGEO              LIBGEO  SNHM14  SNHMC14  SNHMP14  SNHME14  SNHMO14  \
0  01004  AmbÃ©rieu-en-Bugey    13.7     24.2     15.5     10.3     11.2
1  01007            Ambronay    13.5     22.1     14.7     10.7     11.4
2  01014              Arbent    13.5     27.6     15.6     11.1     11.1
3  01024            Attignat    12.9     21.8     14.1     11.0     11.3
4  01025     BÃ¢gÃ©-la-Ville    13.0     22.8     14.1     10.5     11.1

   SNHMF14  SNHMFC14  SNHMFP14    ...      SNHMHO14  SNHM1814  SNHM2614  \
0     11.6      19.1      13.2    ...          11.6      10.5      13.7
1     11.9      19.0      13.3    ...          11.7       9.8      13.8
2     10.9      19.5      11.7    ...          11.8       9.3      13.3
3     11.4      19.0      13.0    ...          11.6       9.6      12.9
4     11.6      19.4      13.6    ...          11.4       9.4      12.8

   SNHM5014  SNHMF1814  SNHMF2614  SNHMF5014  SNHMH1814  SNHMH2614  SNHMH5014
0      16.1        9.7       11.8       12.5       11.0       14.9       18.6
1      14.6        9.2       12.2       12.5       10.2       14.9       16.4
2      16.0        8.9       10.6       12.5        9.6       15.1       18.6
3      14.2        9.3       11.4       12.2        9.7       13.8       15.9
4      15.2        9.0       11.8       12.3        9.7       13.4       16.9

[5 rows x 26 columns]
```

2. Préparation du dataframe « entreprises »

a.

```
entreprises['CODGEO'] = entreprises['CODGEO'].astype(int)
```

b.

```
entreprises['Micro'] = entreprises['E14TS1'] + entreprises['E14TS6']
entreprises['Petite'] = entreprises['E14TS10'] + entreprises['E14TS20']
entreprises['Medium'] = entreprises['E14TS50'] + entreprises['E14TS100']
entreprises['Grande'] = entreprises['E14TS200'] + entreprises['E14TS500']
```

c.

```
entreprises['Somme'] = entreprises['E14TS1'] + entreprises['E14TS6'] +
entreprises['E14TS10'] + entreprises['E14TS20'] + entreprises['E14TS50'] +
entreprises['E14TS100'] + entreprises['E14TS200'] + entreprises['E14TS500']
```

```
entreprises['Micro%'] = entreprises['Micro'] * 100 / entreprises['Somme']
entreprises['Petite%'] = entreprises['Petite'] * 100 / entreprises['Somme']
entreprises['Medium%'] = entreprises['Medium'] * 100 / entreprises['Somme']
entreprises['Grande%'] = entreprises['Grande'] * 100 / entreprises['Somme']
```

d.

```
selection_colonnes = ['CODGEO', 'LIBGEO', 'REG', 'DEP', 'Somme', 'Micro',
'Petite', 'Medium', 'Grande', 'Micro%', 'Petite%', 'Medium%', 'Grande%']

entreprises = entreprises[selection_colonnes]
```

3. Préparation du dataframe « geographie »
On remplace les virgules par des points dans la colonne « longitude ».
On supprime les longitudes inconnues, remplacées par un tiret ou non renseignées.
On assigne le type « float » à la variable « longitude ».

4. Préparation du dataframe « salaires »

```
salaires = salaires[salaires['CODGEO'].apply(lambda x: str(x).isdigit())]
salaires["CODGEO"] = salaires["CODGEO"].astype(int)
```

5. Fusion des dataframe « entreprises » et « geographie »

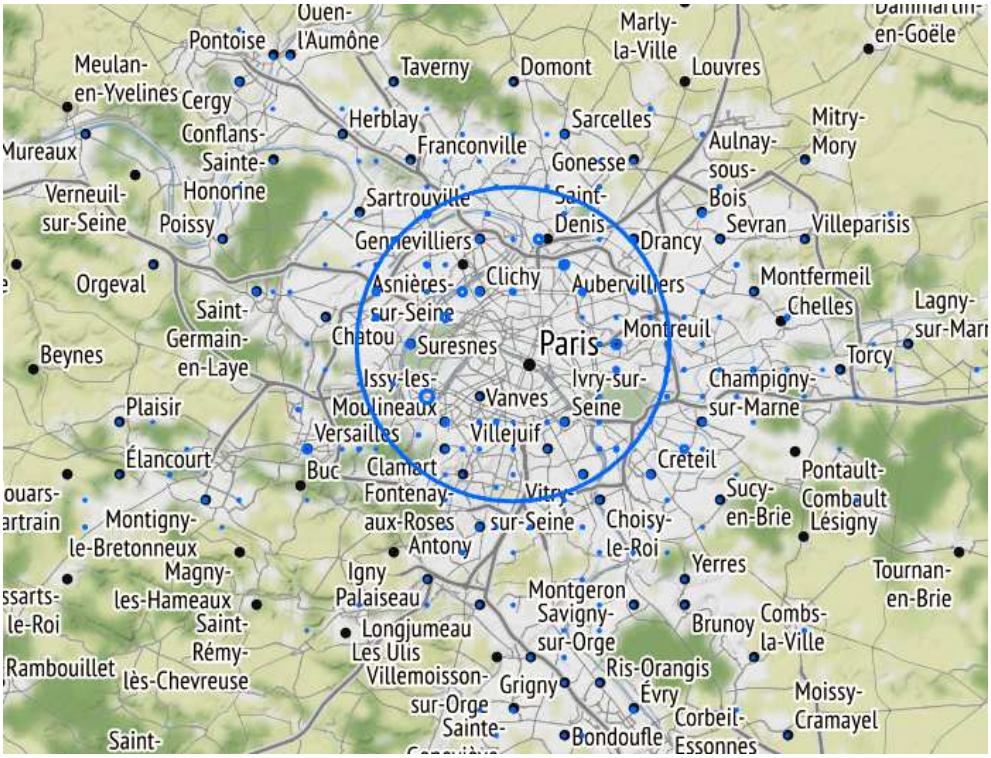
```
>>> data.head(3)
   CODGEO                    LIBGEO  REG DEP  Somme  Micro  Petite  Medium  \
0    1001  L'Abergement-ClÃ©menciat   82  01      3      3       0       0
1    1002     L'Abergement-de-Varey   82  01      1      1       0       0
2    1004        AmbÃ©rieu-en-Bugey   82  01    419    335      70      12

   Grande      Micro%    Petite%   Medium%   Grande%    nom_region  \
0       0  100.000000   0.000000  0.000000  0.000000  RhÃ´ne-Alpes
1       0  100.000000   0.000000  0.000000  0.000000  RhÃ´ne-Alpes
2       2   79.952267  16.706444  2.863962  0.477327  RhÃ´ne-Alpes

  chef.lieu_region               nom_commune  code_insee  latitude  longitude
0             Lyon  L'Abergement-ClÃ©menciat      1001.0     46.15   4.916667
1             Lyon     L'Abergement-de-Varey      1002.0     46.00   5.416667
2             Lyon        AmbÃ©rieu-en-Bugey      1004.0     45.95   5.350000
```

```
>>> data.dtypes
CODGEO                int32
LIBGEO               object
REG                   int64
DEP                  object
Somme                 int64
Micro                 int64
Petite                int64
Medium                int64
Grande                int64
Micro%              float64
Petite%             float64
Medium%             float64
Grande%             float64
nom_region           object
chef.lieu_region     object
nom_commune          object
code_insee          float64
latitude            float64
longitude           float64
dtype: object
```

6. Répartition géographique des entreprises française

a.

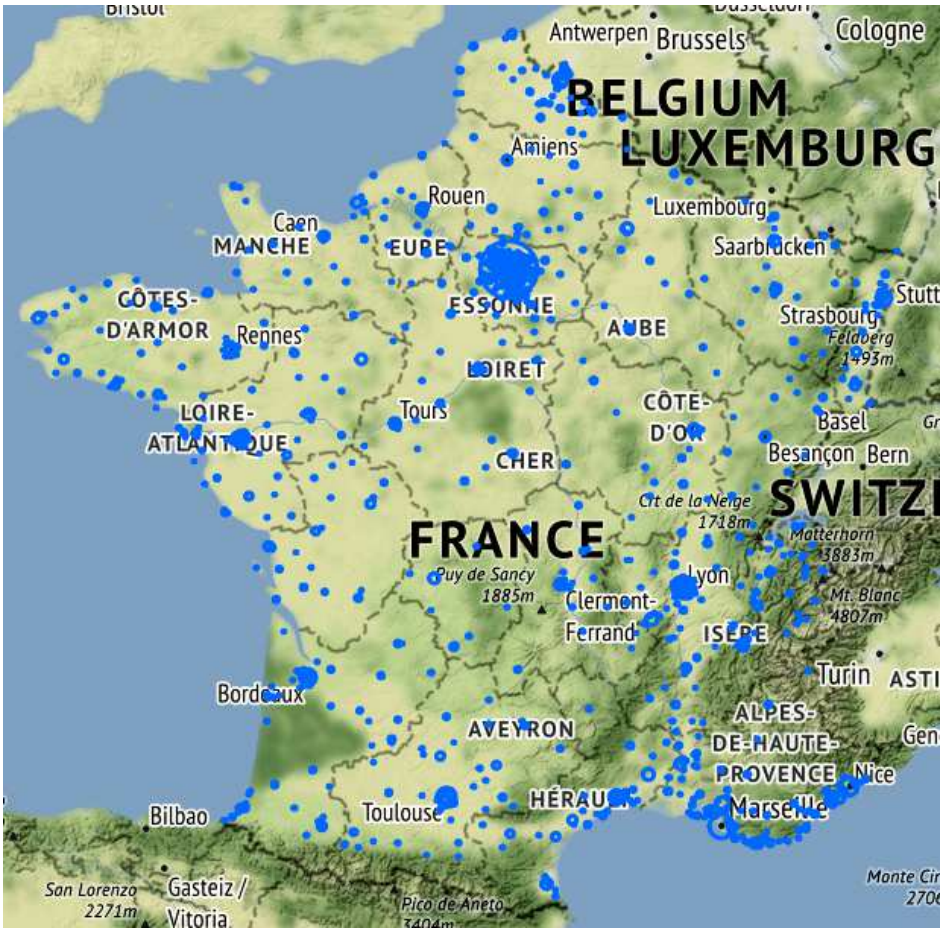
On peut zoomer.

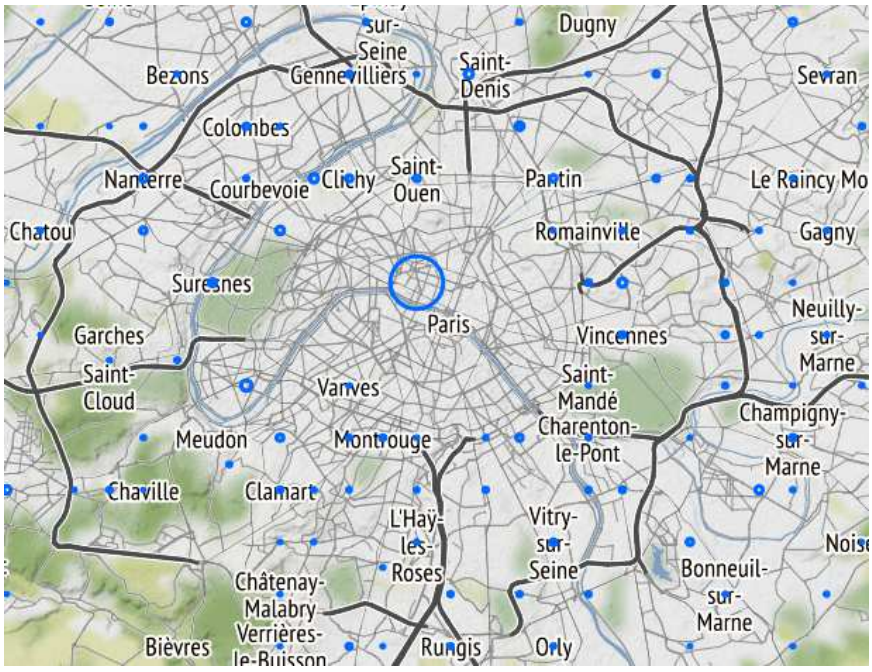


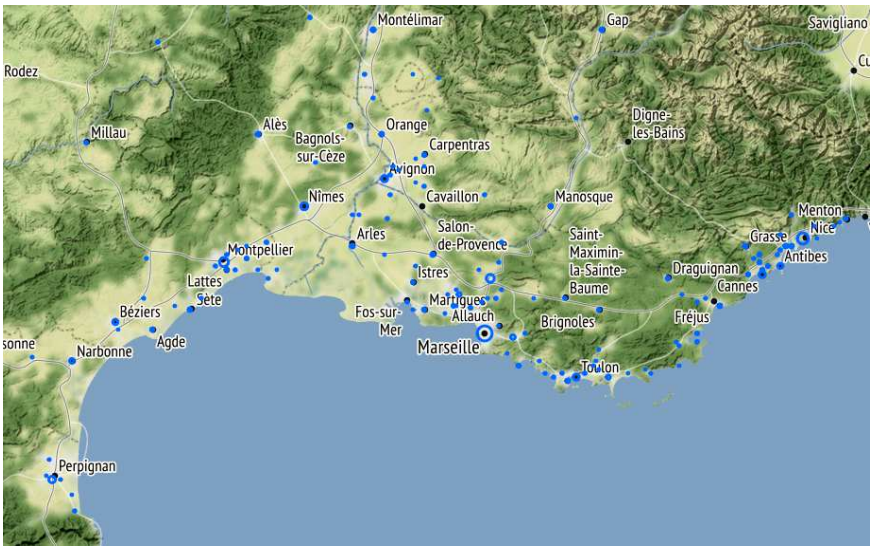
b.

```
45
46  from math import sqrt, pi
47  for k in top1000.index.values :
48      latitude = top1000.loc[k, 'latitude']
49      longitude = top1000.loc[k, 'longitude']
50      rayon = sqrt(top1000.loc[k, 'Somme']/(100*pi))
51      folium.CircleMarker([latitude, longitude], radius =
    rayon).add_to(carte)
52
```

Pour des raisons visuelles, le coefficient 1000 a été remplacé par 100.


Antwerpen
Brussels
Cologne
BELGIUM
LUXEMBURG
Amiens
Rouen
Caen
MANCHE
EURE
Luxembourg
Saarbrücken
CÔTES-
D'ARMOR
Rennes
ESSONNE
AUBE
Strasbourg
Feldberg
1493m
LOIRET
LOIRE-
ATLANTIQUE
Tours
CÔTE-
D'OR
Basel
Besançon
Bern
CHER
SWITZ
Cit de la Neige
1718m
FRANCE
Matterhorn
4383m
Puy de Sancy
1885m
Lyon
Clermont-
Ferrand
Mt. Blanc
4807m
ISÈRE
Turin
Bordeaux
AVEYRON
ALPES-
DE-HAUTE-
PROVENCE
Nice
HÉRAULT
Bilbao
Toulouse
Marseille
San Lorenzo
2271m
Gasteiz /
Vitoria
Pico de Aneto


Dugny
Seine
Bezons
Gennevilliers
Saint-
Denis
Sevran
Colombes
Nanterre
Courbevoie
Clichy
Saint-
Ouen
Pantin
Le Raincy
Chatou
Romainville
Gagny
Suresnes
Paris
Neuilly-
sur-
Marne
Garches
Vincennes
Saint-
Cloud
Vanves
Saint-
Mandé
Champigny-
sur-
Marne
Charenton-
le-Pont
Meudon
Montrouge
Chaville
Clamart
L'Haÿ-
les-
Roses
Vitry-
sur-
Seine
Châtenay-
Malabry
Bonneuil-
sur-
Marne
Bièvres
Verrières-
le-Buisson
Rungis
Orly

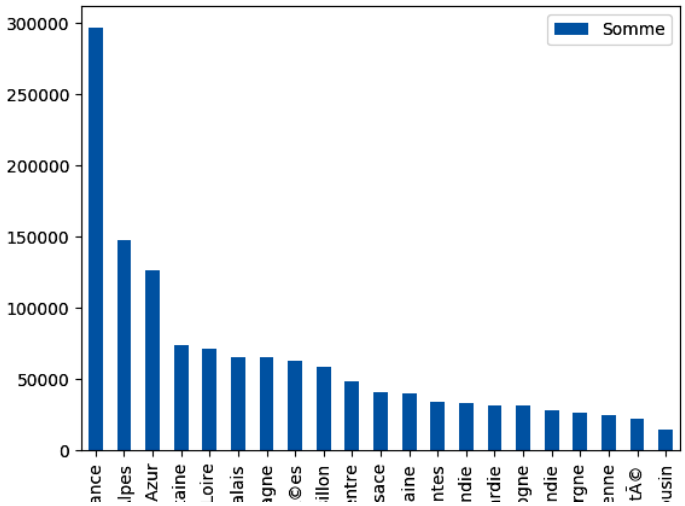


7. Classement des régions les plus porteuses d'emploi

```
regions = data[['nom_region',
'Somme']].groupby('nom_region').sum().sort_values('Somme', ascending = False)
```

```
>>> regions.head(15)
                                Somme
nom_region
Ã鍖le-de-France                 297014
RhÃ´ne-Alpes                   147738
Provence-Alpes-CÃ´te d'Azur    126550
Aquitaine                       74114
Pays de la Loire                71622
Nord-Pas-de-Calais              65681
Bretagne                        65636
Midi-PyrÃ©nÃ©es                 63345
Languedoc-Roussillon            59056
Centre                          49038
Alsace                          40766
Lorraine                        40321
Poitou-Charentes                34491
Haute-Normandie                 33009
Picardie                        31571
```

```
regions.plot.bar()
```




```
>>> regions.describe()
              Somme
count      21.000000
mean    64225.428571
std     62849.972968
min     15106.000000
25%     31412.000000
50%     40766.000000
75%     65681.000000
max    297014.000000
```

Le nombre moyen d'entreprises par région est 64 225 mais la moitié des régions comptent moins de 40 766 entreprise. La moyenne est ainsi très supérieure à la médiane. Elle est tirée vers le haut par les trois régions les plus porteuses, en particulier la région Ile-de-France.

8. Ecarts de salaire selon le lieu et le genre

a.

```
salaires.sort_values(by = 'SNHM14', ascending = False).head(10)['LIBGEO']
```

```
>>> salaires.sort_values(by = 'SNHM14', ascending
= False).head(10)['LIBGEO']
4225    Saint-Nom-la-BretÃ¨che
4170             Feucherolles
4173                Fourqueux
4872        Neuilly-sur-Seine
4237              Le VÃ©sinet
4194             Mareil-Marly
4156               Chambourcy
4169        L'Ã©tang-la-Ville
4881              Vaucresson
4164        Croissy-sur-Seine
Name: LIBGEO, dtype: object
```

```
salaires.sort_values(by = 'SNHM14', ascending = False).tail(10)['LIBGEO']
```

```
>>> salaires.sort_values(by = 'SNHM14', ascending
= False).tail(10)['LIBGEO']
230                    Ruoms
1176                Le Vigan
5132                 Salazie
1137          La Grand-Combe
5135                  Cilaos
763                 Aubusson
4314         Blaye-les-Mines
604           Saint-Savinien
2237     Miramont-de-Guyenne
1184     BagnÃ¨res-de-Luchon
Name: LIBGEO, dtype: object
```

Remarque :
On ne dispose pas de la population de chaque commune, ce qui ne permet pas d'obtenir des moyennes pondérées.

b.
On peut étudier une ville en particulier ou rechercher où les écarts de salaires entre genres sont les plus grands.

```
salaires_Paris = salaires[salaires.LIBGEO == 'Paris']
>>> salaires_Paris['SNHMF14']
3923    19.1
Name: SNHMF14, dtype: float64

>>> salaires_Paris['SNHMH14']
3923    24.7
Name: SNHMH14, dtype: float64
```

A Paris en 2014, le salaire net horaire moyen des femmes est 19,1 € et celui des hommes est 24,7 €.

Création de nouvelles colonnes calculant les différences de salaire hommes-femmes :

```
salaires['diff_moyen'] = salaires['SNHMH14'] - salaires['SNHMF14']
salaires['diff_cadres_sup'] = salaires['SNHMHC14'] - salaires['SNHMFC14']
salaires['diff_profs_inter'] = salaires['SNHMHP14'] - salaires['SNHMFP14']
salaires['diff_employes'] = salaires['SNHMHE14'] - salaires['SNHMFE14']
salaires['diff_ouvriers'] = salaires['SNHMHO14'] - salaires['SNHMFO14']
```

```
>>> salaires['diff_moyen'].describe()
count    5107.000000
mean        2.813198
std         1.661317
min        -0.200000
25%         1.900000
50%         2.500000
75%         3.300000
max        28.000000
Name: diff_moyen, dtype: float64
```

La différence de salaire net horaire moyen par commune va de 28 € (en faveur des hommes) à – 0,20 € (différence en faveur des femmes). La différence de salaire moyenne, pour les différentes communes, est 2,80 € de l'heure. Dans 50 % des communes la différence est supérieure ou égale à 2,50 €.

Effectuons un tri selon les différences de salaire.

```
tri1 = salaires.sort_values(by = 'diff_moyen', ascending = False)
tri2 = salaires.sort_values(by = 'diff_cadres_sup', ascending = False)
tri3 = salaires.sort_values(by = 'diff_profs_inter', ascending = False)
tri4 = salaires.sort_values(by = 'diff_employes', ascending = False)
tri4 = salaires.sort_values(by = 'ouvriers', ascending = False)
```

```
>>> tri1[['LIBGEO', 'diff_moyen']].head(10)
                        LIBGEO  diff_moyen
4225     Saint-Nom-la-BretÃ¨che        28.0
4170                Feucherolles        25.0
4173                   Fourqueux        23.5
4156                  Chambourcy        23.4
4872           Neuilly-sur-Seine        19.0
4237                 Le VÃ©sinet        18.9
4194                Mareil-Marly        17.0
4164           Croissy-sur-Seine        16.4
3628  Saint-Didier-au-Mont-d'Or        16.3
4209                Noisy-le-Roi        16.2
```

Dans les villes où la différence hommes-femmes est la plus importante, on retrouve des communes privilégiées.

```
>>> tri1[['LIBGEO', 'diff_moyen']].tail(10)
                      LIBGEO  diff_moyen
5061              Grand-Bourg         0.3
748                 Rostrenen         0.3
4884            Aubervilliers         0.3
1176                 Le Vigan         0.3
4890               Le Bourget         0.2
5111  Saint-Laurent-du-Maroni         0.2
3849                Beaumont         0.1
786                  Nontron         0.1
5059                Gourbeyre        -0.2
5112  MontsinÃ©ry-Tonnegrande        -0.2
```

Dans les communes où la différence hommes-femmes est la plus faible, on retrouve des villes plus défavorisées.

```
>>> salaires['diff_cadres_sup'].describe()
count    5107.000000
mean        4.978774
std         2.649876
min       -10.000000
25%         3.500000
50%         4.800000
75%         6.200000
max        28.500000
Name: diff_cadres_sup, dtype: float64
```

Les différences de salaires hommes-femmes selon les communes sont en moyenne plus importantes pour les cadres supérieurs : environ 5 € de l'heure, mais les salaires sont plus élevés. La dispersion selon les communes est plus grande, allant de – 10€ à 28,5 €, mais le nombre de cadres supérieurs dans certaines communes est sans doute très réduit.

Les différences de salaires hommes-femmes à Paris selon les catégories d'emplois :

```
>>> salaires_Paris = salaires[salaires.LIBGEO == 'Paris']

>>> salaires_Paris[['diff_moyen', 'diff_cadres_sup', 'diff_profs_inte
r', 'diff_employes', 'diff_ouvriers']]
      diff_moyen  diff_cadres_sup  diff_profs_inter  diff_employes  \
3923         5.6              9.1               3.0            0.3

      diff_ouvriers
3923            0.8
```

La différence de salaire la plus faible se trouve chez les employés.

On peut calculer les différences de salaires relatives en pourcentage du salaire moyen.

```
salaires['diff_moyen%'] = salaires['diff_moyen'] * 100 / salaires['SNHM14']
salaires['diff_cadres_sup%'] = salaires['diff_cadres_sup'] * 100 /
salaires['SNHMC14']
salaires['diff_profs_inter%'] = salaires['diff_profs_inter'] * 100 /
salaires['SNHMP14']
salaires['diff_employes%'] = salaires['diff_employes'] * 100 /
salaires['SNHME14']
salaires['diff_ouvriers%'] = salaires['diff_ouvriers'] * 100 /
salaires['SNHMO14']

>>> salaires_Paris[['diff_moyen%', 'diff_cadres_sup%', 'diff_profs_inter%
', 'diff_employes%', 'diff_ouvriers%']]
      diff_moyen%  diff_cadres_sup%  diff_profs_inter%  diff_employes%  \
3923    25.225225         28.526646           17.44186        2.439024

      diff_ouvriers%
3923        6.060606
```

La différence de salaire moyen hommes-femmes à Paris est de 25 % du salaire moyen et va de 29 % pour les cadres supérieurs à 2 % pour les employés.

Prolongement : on peut observer les différences de salaires hommes-femmes selon les âges et constater qu'elles augmentent avec la catégorie d'âge.

12. Paradise Papers

1. Etude des pays impliqués dans les « Paradise Papers »

a.

```
entites.shape
>>> entites.shape
(24957, 18)
```

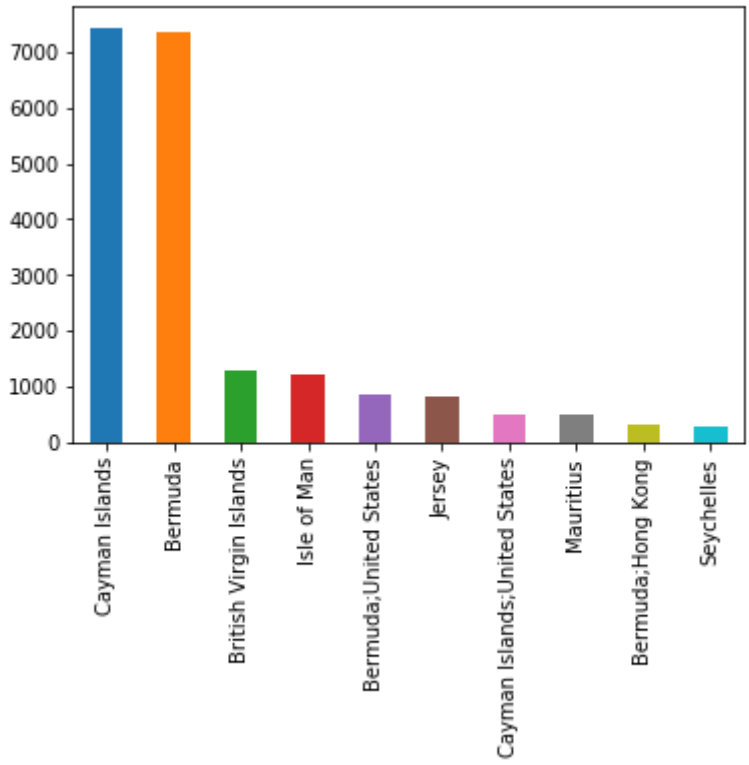
Le fichier des entités comporte 24 957 lignes et 18 variables.
Les noms des différentes variables sont les suivants :

```
entites.columns
>>> entites.columns
Index(['labels(n)', 'n.valid_until', 'n.country_codes', 'n.countries',
       'n.node_id', 'n.sourceID', 'n.address', 'n.name',
       'n.jurisdiction_description', 'n.service_provider', 'n.jurisdiction',
       'n.closed_date', 'n.incorporation_date', 'n.ibcRUC', 'n.type',
       'n.status', 'n.company_type', 'n.note'],
      dtype='object')
```

```
paradis = entites['n.countries'].value_counts()
```

```
>>> paradis[:20]
Cayman Islands                        7440
Bermuda                               7351
British Virgin Islands                1299
Isle of Man                           1219
Bermuda;United States                  839
Jersey                                 818
Cayman Islands;United States           502
Mauritius                              479
Bermuda;Hong Kong                      313
Seychelles                             285
Bermuda;United Kingdom                 192
Cayman Islands;Hong Kong               179
United States                          166
Cayman Islands;China                   132
British Virgin Islands;Jersey          131
British Virgin Islands;China           121
British Virgin Islands;Hong Kong       105
China;Seychelles                        97
Isle of Man;United Kingdom              88
Bermuda;Switzerland                     87
Name: n.countries, dtype: int64
```

```
paradis[:10].plot.bar()
plt.show()
```



Pour l'essentiel, les paradis fiscaux représentés dans les « Paradise Papers » sont les îles Cayman et les Bermudes.

b.

```
executeurs.shape
(77012, 18)
```

Le fichier comporte 77 012 lignes et 18 variables dont les noms sont les suivants.

```
executeurs.columns
Index(['labels(n)', 'n.valid_until', 'n.country_codes', 'n.countries',
       'n.node_id', 'n.sourceID', 'n.address', 'n.name',
       'n.jurisdiction_description', 'n.service_provider', 'n.jurisdiction',
       'n.closed_date', 'n.incorporation_date', 'n.ibcRUC', 'n.type',
       'n.status', 'n.company_type', 'n.note'],
      dtype='object')
```

Remarque : on peut constater que la reine d'Angleterre est citée. 'The Duchy of Lancaster' est le domaine privé de la souveraine britannique.

```
executeurs[executeurs['n.name'] == 'The Duchy of Lancaster']
```

```
>>> executeurs[executeurs['n.name'] == 'The Duchy of Lancaster']
      labels(n)                          n.valid_until n.country_codes  \
1185  ["Officer"]  Appleby data is current through 2014             GBR

     n.countries  n.node_id                n.sourceID  n.address  \
1185         NaN   84100003  Paradise Papers - Appleby        NaN

                    n.name  n.jurisdiction_description  n.service_provider
\
1185  The Duchy of Lancaster                         NaN                 NaN
```

On constitue un dataframe comptant les pays exécuteurs. Puis on affiche les 20 pays les plus souvent cités.

```
pays_executeurs = executeurs['n.countries'].value_counts()
>>> pays_executeurs[:20]
United States                  17303
United Kingdom                  4298
Hong Kong                       3262
China                           3050
Bermuda                         2955
Cayman Islands                  1925
Bermuda;United Kingdom          1362
Canada                          1226
China;Hong Kong                  981
Switzerland                      969
British Virgin Islands           919
Singapore                        789
Jersey                           651
Taiwan                           645
Australia                        640
Japan                            570
Isle of Man;United Kingdom       508
Ireland                          497
Isle of Man                      489
France                           422
Name: n.countries, dtype: int64
```

Ces données montrent à qui profitent les montages financiers. En premier lieu, on trouve les Etats-Unis. La France apparaît en vingtième position des pays exécuteurs les plus cités.

c.

```
adresses.shape
>>> adresses.shape
(59228, 18)
```

Le fichier comporte 59 228 lignes et 18 variables dont les noms sont les suivants.

```
adresses.columns
```

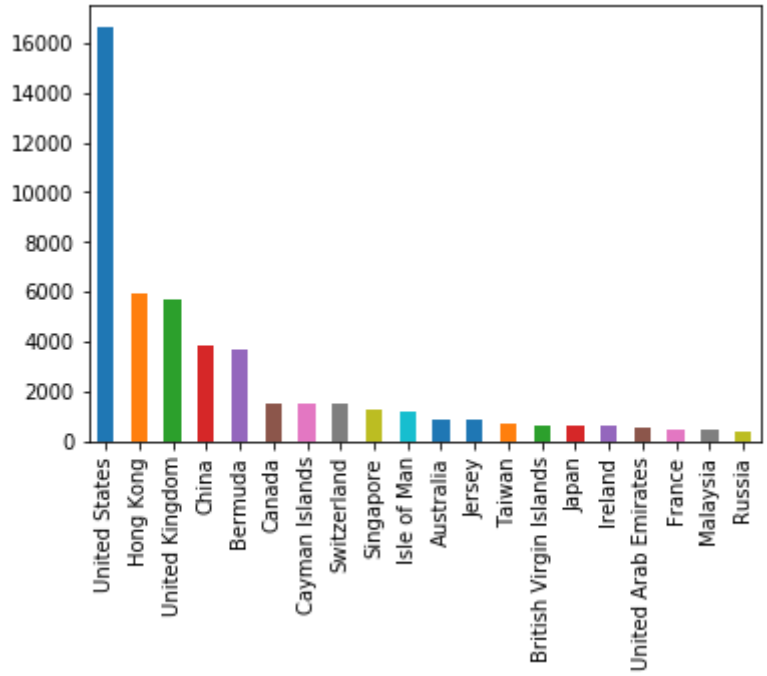
```
>>> adresses.columns
Index(['labels(n)', 'n.valid_until', 'n.country_codes', 'n.countries',
       'n.node_id', 'n.sourceID', 'n.address', 'n.name',
       'n.jurisdiction_description', 'n.service_provider', 'n.jurisdiction',
       'n.closed_date', 'n.incorporation_date', 'n.ibcRUC', 'n.type',
       'n.status', 'n.company_type', 'n.note'],
      dtype='object')
```

On constitue un dataframe comptant le nombre de fois où les pays des « clients » sont cités.

```
pays_clients = adresses['n.countries'].value_counts()
```

```
>>> pays_clients[:20]
United States              16631
Hong Kong                   5963
United Kingdom              5730
China                       3875
Bermuda                     3668
Canada                      1533
Cayman Islands              1519
Switzerland                 1495
Singapore                   1316
Isle of Man                 1172
Australia                    863
Jersey                       851
Taiwan                       716
British Virgin Islands       632
Japan                        620
Ireland                      603
United Arab Emirates         525
France                       505
Malaysia                     469
Russia                       428
Name: n.countries, dtype: int64
```

La France apparaît dans le top 20 des pays des pays où des « clients » des Paradise Papers sont localisés (mais très loin derrière les Etats-Unis en effectif).



Les adresses en France :

```
adresses[adresses['n.countries'] == 'France']
```

```
2615                                Tour Societe Generale
```

A la ligne 2615 est citée la Société Générale.

2. Représentation sur une carte des principaux pays impliqués

a.

```
>>> geographie.head(5)
                    state  latitude  longitude
0               Australia     -37.8      145.0
1                 Bermuda      32.3      -64.8
2  British Virgin Islands      18.4      -64.6
3                  Canada      45.4      -75.7
4          Cayman Islands      19.3      -81.3

>>> geographie.dtypes
state          object
latitude      float64
longitude     float64
dtype: object

>>> geographie.shape
(22, 3)
```

b.

Les nouvelles colonnes indiquent, pour chaque pays, le nombre de fois où le pays est cité comme paradis fiscal, comme pays d'un exécuteur ou comme pays d'un client. La dernière

colonne, nommée poids, fait la somme des trois précédentes et compte donc le nombre de fois que le pays est cité dans les Paradise Papers.

```
>>> geographie.head(5)
                    state  latitude  longitude  nb paradis  nb executeurs  \
0               Australia     -37.8      145.0          10            640
1                 Bermuda      32.3      -64.8        7351           2955
2  British Virgin Islands      18.4      -64.6        1299            919
3                  Canada      45.4      -75.7          14           1226
4          Cayman Islands      19.3      -81.3        7440           1925

   nb clients  poids
0         863   1513
1        3668  13974
2         632   2850
3        1533   2773
4        1519  10884
```

c.

On souhaite que l'aire de chaque disque soit égale à un dixième du poids du pays (le coefficient un dixième est choisi pour des raisons de taille du disque sur la carte).

$\pi r^2 = \frac{1}{10} p$ donne $r = \sqrt{\frac{p}{10\pi}}$. D'où l'expression à la ligne 51.

3. Représentation des liens sur une carte

a.

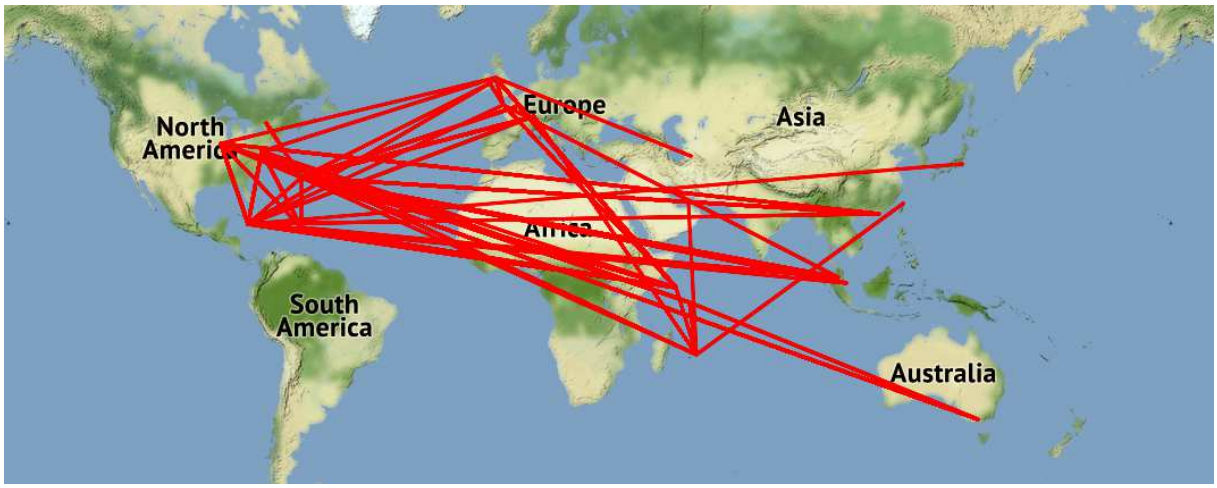
```
>>> liens.columns
Index(['node_1', 'rel_type', 'node_2', 'r.sourceID', 'r.valid_until',
       'r.start_date', 'r.end_date'],
      dtype='object')

>>> liens.shape
(364456, 7)

>>> liens.head(5)
     node_1           rel_type    node_2                r.sourceID  \
0  82019024  registered_address  81027146  Paradise Papers - Appleby
1  82019039  registered_address  81027146  Paradise Papers - Appleby
2  82019028  registered_address  81027146  Paradise Papers - Appleby
3  82019060  registered_address  81027146  Paradise Papers - Appleby
4  82019037  registered_address  81027146  Paradise Papers - Appleby

                        r.valid_until r.start_date r.end_date
0  Appleby data is current through 2014          NaN        NaN
1  Appleby data is current through 2014          NaN        NaN
2  Appleby data is current through 2014          NaN        NaN
3  Appleby data is current through 2014          NaN        NaN
4  Appleby data is current through 2014          NaN        NaN
```

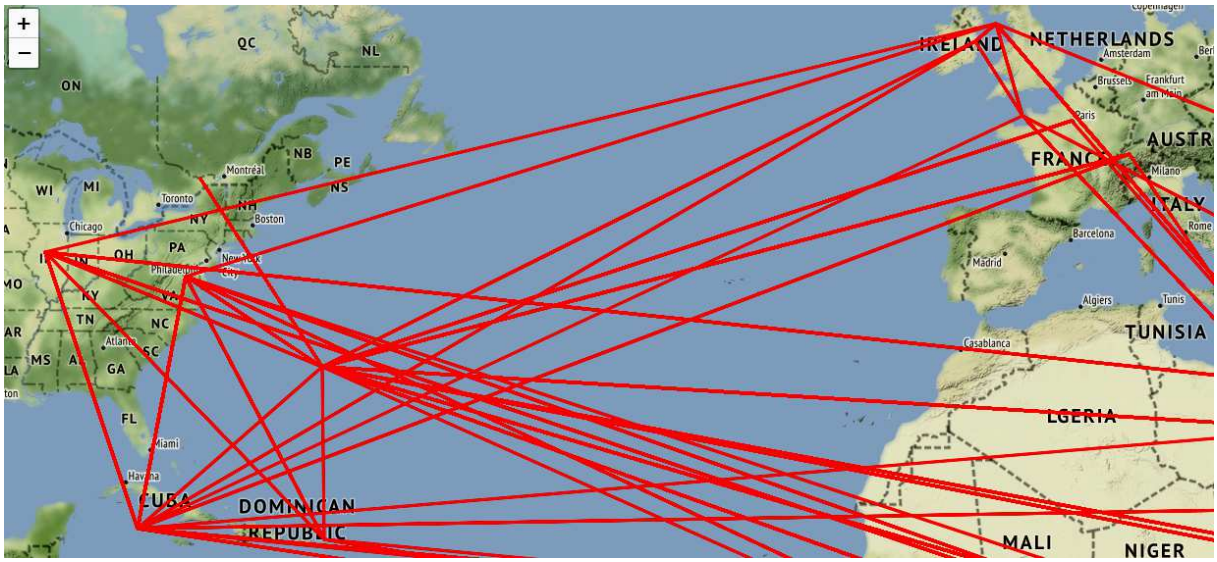
b.

```
# 3. Representation des liens sur une carte
carte_liens = folium.Map(location=[0,0],zoom_start = 2,tiles='Stamen Terrain')

for k in range(10000):
    # On cherche dans les 3 fichiers executeurs, adresses
    # et entites, le pays correspondant au noeud 1
    # reset_index(drop = True) permet de reinitialiser l'index
    noeud1 = liens.loc[k,'node_1']
    try :
        if executeurs[executeurs['n.node_id'] == noeud1]['n.countries'].empty == False:
            pays1 = executeurs[executeurs['n.node_id'] == noeud1]['n.countries'].reset_index(drop = True)
        elif adresses[adresses['n.node_id'] == noeud1]['n.countries'].empty == False:
            pays1 = adresses[adresses['n.node_id'] == noeud1]['n.countries'].reset_index(drop = True)
        elif entites[entites['n.node_id'] == noeud1]['n.countries'].empty == False:
            pays1 = entites[entites['n.node_id'] == noeud1]['n.countries'].reset_index(drop = True)
        pays1 = pays1[0]
    except:
        pays1 = ''
    # si le pays 1 est dans le fichier geographie, on cherche
    # dans les 3 fichiers le pays correspondant au noeud 2
    if geographie[geographie['state']== pays1].empty == False:
        noeud2 = liens.loc[k,'node_2']
        try:
            if executeurs[executeurs['n.node_id'] == noeud2]['n.countries'].empty == False:
                pays2 = executeurs[executeurs['n.node_id'] == noeud2]['n.countries'].reset_index(drop = True)
            elif adresses[adresses['n.node_id'] == noeud2]['n.countries'].empty == False:
                pays2 = adresses[adresses['n.node_id'] == noeud2]['n.countries'].reset_index(drop = True)
            elif entites[entites['n.node_id'] == noeud2]['n.countries'].empty == False:
                pays2 = entites[entites['n.node_id'] == noeud2]['n.countries'].reset_index(drop = True)
            pays2 = pays2[0]
        except:
            pays2 = ''
```

```
88      # Si les pays 1 et 2 sont dans le fichier geographie, on
89      # recupere leur latitude et longitude et on les relie
90      if geographie[geographie['state']== pays1].empty == False:
91          if geographie[geographie['state']== pays2].empty == False:
92              pays = geographie[geographie['state']==
    pays1].reset_index(drop = True)
93              lat1 = pays['latitude'][0]
94              long1 = pays['longitude'][0]
95              pays = geographie[geographie['state']==
    pays2].reset_index(drop = True)
96              lat2 = pays['latitude'][0]
97              long2 = pays['longitude'][0]
98              folium.PolyLine([[lat1,long1],[lat2,long2]],
    color='red').add_to(carte_liens)
99
100 carte_liens.save('C:/Users/dutar/Desktop/carte_liens.html')
```

Aspect de la carte obtenue :





Références

BERHOUET J., DUTARTE P., GLEBA F. (2017), *Statistique, probabilités et jugement critique* – actes du séminaire académique Sciences et jugement critique – académie de Créteil 2017 maths.ac-creteil.fr.

BERNARD A., EHRHARDT C. (2017), *Les lois du hasard : enjeux mathématiques, historiques, citoyens* – actes du séminaire national de l'ARDM.

BROCH H., CHARPAK G. (2002), *Devenez sorciers, devenez savants*.

CORTECS (COllectif de Recherche Transdisciplinaire Esprit Critique & Sciences) est un collectif né en 2010 à Grenoble, Marseille et Montpellier. Son but est de réunir tous les acteurs, enseignants, chercheurs, étudiants travaillant sur un sujet développant l'esprit critique, quelle que soit leur origine disciplinaire.
http://cortecs.org/thematix/mathematiques/

DELAHAYE J.-P., GAUVRIT N. (2012), *Comme par hasard !* – book-e-book 2012.

DROESBEKE J.-J., VERMANDELE C. (2016), *Les nombres au quotidien, leur histoire, leurs usages* Technip 2016.

DROESBEKE J.-J., VERMANDELE C. (2018), *Histoire(s) de(s) données numériques* EDP Sciences.

DUTARTE, P., DELZONGLE, F., MAATI, H., CARDINAL, J.-P., COUPRY, A., & DHERISSARD, S. (2007). *Statistique et citoyenneté, le citoyen face au chiffre*. Brochure 135 de l'IREM de Paris Nord.

DUTARTE, P. (2011). *Évolution de la pratique statistique dans l'enseignement du second degré en France. Statistique et Enseignement* 2(1), sept. 2011, 31-42.

EDUSCOL (2016) :
http://eduscol.education.fr/cid107295/former-l-esprit-critique-des-eleves.html

GAUVRIT N. (2007) – *Statistiques : méfiez-vous !* – Ellipse.

ZEISEL Hans, KAYE D. H. et D. (2006), *Prove It with Figures (Statistics for Social Science and Behavioural Sciences)* - Springer.