BUT1 – OUTILS NUMÉRIQUES POUR LES STATISTIQUES DESCRIPTIVES TRAVAUX PRATIQUES

IUT DE NANTES – DÉPARTEMENT INFORMATIQUE – 2021/2022

Table des matières

Le but de ces TP est notamment de vous faire découvrir un certain nombre de librairies `Python` utiles pour la visualisation et le traitement de données en statistiques. On s'appuiera notamment sur la librairie `Matplotlib` mais pas que.

Organisation du temps de travail

Première semaine :

- TD1 (1h20) : cours sur les statistiques descriptives – premiers indicateurs numériques
- TP1 (2h40) : visualisation de données

Deuxième semaine :

- TD2 (1h20) : cours sur les statistiques à deux variables
- TP2 (2h40) : statistiques à deux variables

Troisième semaine :

- TD3 (1h20) : séance de transition
- TP3 (2h40) : analyse fréquentielle de chiffrement

Quatrième semaine :

- TP4 (2h40) : visualisation de données géographiques (Sae)
- TD4 (1h20) : *évaluation terminale sur machine*

Au total : 4 créneaux de TD et 8 créneaux de TP par étudiant.

Les trois premiers TPs seront à rendre en binôme à la fin de chaque semaine sur MADOC. Évalués, ils permettront de vous fournir une première note.

TP 1 – Visualiser des données

Ces premiers exercices ont pour but de vous familiariser avec certaines commandes Python pour afficher et sauvegarder les graphiques obtenues à partir de données statistiques. [1]

Exercice 1.1 – Consommation des ménages – Le but de cet exercice est de faire des premières visualisations de données. Vous allez utiliser les données présentes dans le fichier suivant :

`consommation_menages.csv` .

Afin d'accéder aux données contenues dans ce fichier, vous pourrez utiliser les commandes suivantes :

```
import pandas as pd
data = pd.read_csv('consommation_menages.csv', sep=',')
```

1. Que renvoient les commandes suivantes :

```
type(data)
data.info
data.Fonction
data.1960
data['Fonction']
data['1960]
data[['1960','2020']]
```

N'hésitez pas à aller consulter la documentation de la librairie `pandas`.

1. Représenter par un diagramme en barres verticales le contenu de la colonne `'1960'`.
2. Représenter par un diagramme camembert le contenu de la colonne `'2020'` .
3. Représenter par un diagramme en barres verticales les données correspondantes aux années 1960 et 2020. Vous devriez faire afficher le diagramme présenté en Figure 1.

Exercice 1.2 – Temps de vie de cartes mères – Un fabriquant de cartes mères teste la durée de vie de son produit phare. Les durée de vie (temps écoulé entre la première mise en tension et la première panne) d'un échantillon des cartes produites sont fournies dans le fichier

`temps_de_vie.csv` .

En utilisant la librairie `pandas` pour lire ce fichier, et la fonction `plt.hist`, faites afficher un histogramme de la répartition des temps de vie de cet échantillon.

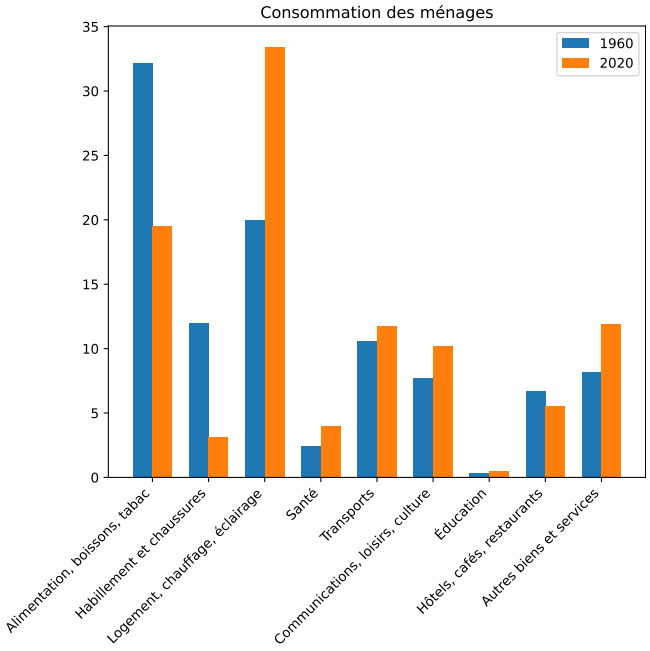
Exercice 1.3 – Pyramide des âges – Le but de cet exercice est de construire la pyramide des âges française. Pour cela, on utilisera le fichier

`pyramide_ages.csv`

dont les données proviennent du site de l'Insee. On l'ouvrira avec la suite de commandes suivantes :

1. L'auteur remercie Mme Fontaine (Insee) de lui avoir fourni la grande partie des données Insee qui serviront pour l'ensemble de ces exercices.

FIGURE 1




```
import pandas as pd
data = pd.read_csv('pyramide_ages.csv', sep=',')
```

1. Que renvoient les commandes suivantes :

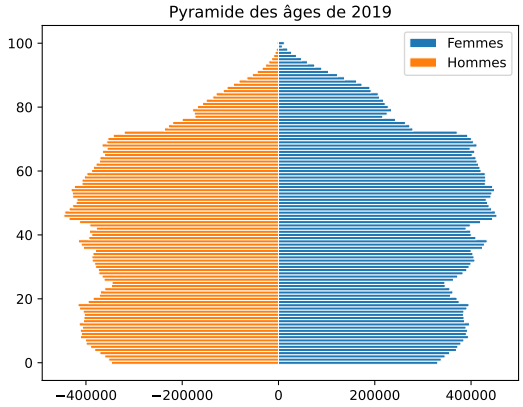
```
data.columns
data['age']
data[data['age']<60]
```

On cherche à représenter une pyramide des âges construite à partir de de barres horizontales.

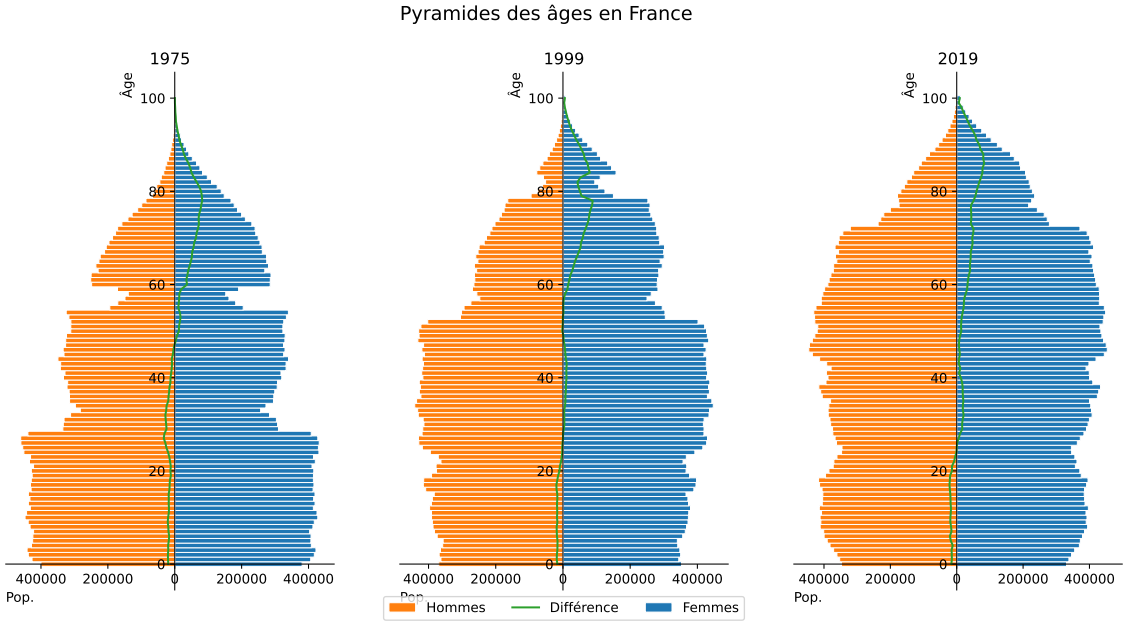
2. Pour représenter des données sous forme de barres horizontales, on utilise la commande `ax.barh`. Afin de comprendre son fonctionnement, on regardera l'effet des commandes suivantes :

```
import matplotlib.pyplot as plt
# On initialise notre figure
fig, axes = plt.subplots(1,1)
# On trace deux barres horizontales
axes.barh(y=1, width = 3, height = 0.5)
axes.barh(y=2, width = 1, height = 0.5)
plt.show()
```

3. En utilisant la méthode `barh`, afficher la pyramide des âges suivante.



4. En utilisant les données présentes dans ce fichier, afficher le graphique suivant, où la courbe verte correspond à la différence des effectifs.



Exercice 1.4 – Mobilité sociale – Statistiques qualitatives croisées – Le but de cet exercice est de construire deux graphiques illustrant la destinée sociale d'individus en fonction de la catégorie sociale de leur père. Vous devrez utiliser les fichiers suivants

`mobilite_sociale_Insee.ods` et `TP1_mobilite_sociale.py`

le premier étant le fichier source contenant les données et le second étant un fichier `.py` dans lequel les données ont déjà été mises en forme. Les données du premier fichier sont disponibles à l'adresse suivante :

`https://www.insee.fr/fr/statistiques/4797592?sommaire=4928952` .

On cherche à représenter ces données sous forme de diagrammes en barres horizontales. On va utiliser de nouveau la méthode `barh` de la librairie `matplotlib.pyplot` ;

1. La méthode `.barh` est assez permissive : on peut lui faire prendre des chaînes de caractères en la variable y. Constatons cela avec les commandes suivantes. À quoi sert l'argument `left` ?

```python
import matplotlib.pyplot as plt
# On initialise notre figure
fig, ax = plt.subplots()
# On trace des barres horizontales
ax.barh(y='A', width = 3, left= 0, height = 0.5)
ax.barh(y='A', width = 2, left= 3, height = 0.5)
ax.barh(y='B', width = 1, left= 0, height = 0.5)
ax.barh(y='B', width = 4, left= 1, height = 0.5)
plt.show()
```

2. Reprenez les lignes ci-dessus et ajoutez les lignes suivantes. Que se passe-t-il ?

```python
import matplotlib.pyplot as plt
# On initialise notre figure
fig, ax = plt.subplots()
# On trace des barres horizontales
ax.barh(y='A', width = 3, left= 0, height = 0.5)
ax.barh(y='A', width = 2, left= 3, height = 0.5)
ax.barh(y='B', width = 1, left= 0, height = 0.5)
ax.barh(y='B', width = 4, left= 1, height = 0.5)
# On ameliore l'affichage
ax.set_ylim(-1,1)
ax.xaxis.set_visible(False)
plt.show()
```

3. Produisez le graphique suivant en vous servant des données ci-dessous.

```python
agriculteur =  [27.6, 9.0, 12.2, 14.8, 26.0, 10.4]
category_colors = ["tab:olive","tab:purple",
    "tab:blue","tab:cyan", "tab:brown","tab:orange"]
```

Vous pourrez notamment :

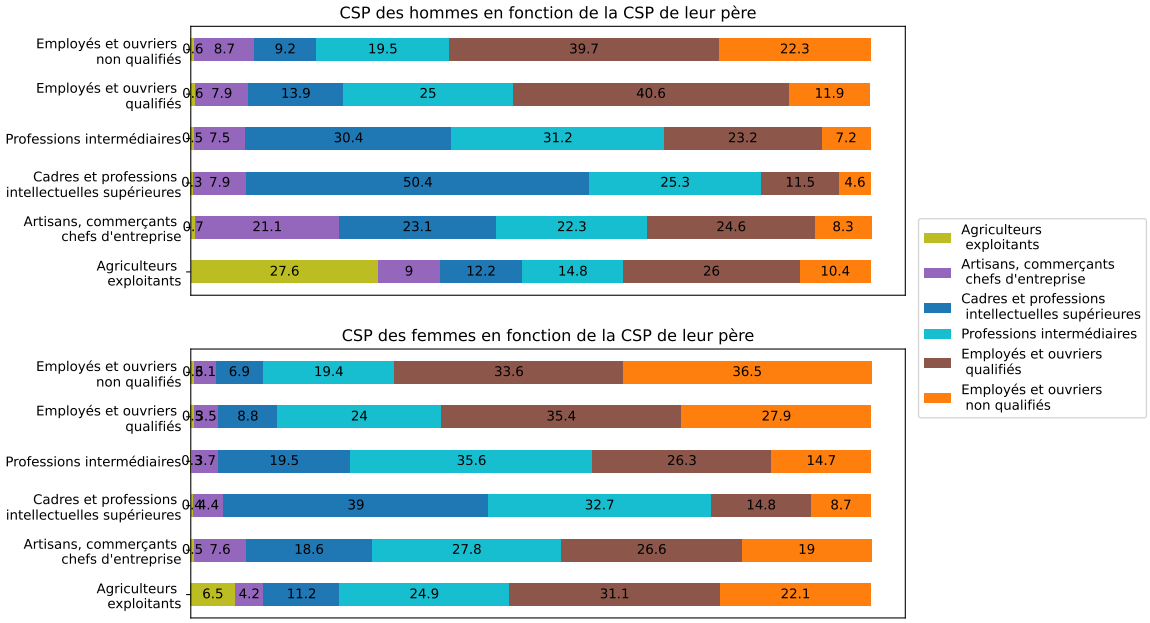
— placer la ligne de commande `plt.tight_layout()` juste avant l'affichage du graphique afin d'optimiser notre affichage ;

— utiliser les arguments optionnels de la méthode `ax.legend()` ;

— forcer la taille de la figure avec l'argument optionnel `figsize` de la méthode `plt.subplots()`.

Vous devriez arriver à un graphique comme suit :

4. Utiliser les données complètes, fournies dans le début du TP, afin de générer le diagramme suivant.



TP 2 – Série statistique à deux variables

Exercice 2.1 – Premier exemple de correlation linéaire – On considère X et Y deux listes de nombres de même taille.

1. *Quelques fonctions préliminaires :*

   a) Écrire une fonction `point_moyen(X,Y)` qui renvoie les coordonnées du point moyen des séries statistiques X et Y.

   b) Écrire une fonction `coeff_corr(X,Y)` qui renvoie le coefficient de corrélation des séries statistiques X et Y.

   c) Écrire une fonction `coeffs_droite_reg(X,Y)` qui renvoie les coefficients a et b de la droite de régression linéaire (d'équation $y = ax + b$) des séries statistiques X et Y.

   d) Écrire une fonction `etude(X,Y)` qui affiche un graphique contenant :

      — les points représentant la série statistique (X, Y) ;

      — le point moyen ;

      — la droite de régression linéaire associée ;

      — faisant apparaître le coefficient de corrélation.

   En utilisant les données suivantes, vous devriez obtenir le graphique présenté en Figure 2.

Année	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002
PIB	1121.0	1132.2	1149.1	1138.9	1162.4	1181.8	1194.9	1217.6	1259.1	1299.5	1348.8	1377.1	1393.7
Dépense	627.5	631.7	637.5	633.7	641.2	649.0	657.3	658.2	680.7	702.6	721.2	740.1	748.9

2. *Un exemple concret :* On fournit dans le fichier

   `PercenRevParlVSEtude.csv`

   (source : http://piketty.pse.ens.fr/fr/ideologie) deux séries statistiques prises sur un échantillon représentatif de la population américaine (référence : WIR 2018, Figure 5.4.1. College attendance rates and parent income rank in the US (Chetty-Saez, Equal opportunity project)). La première est le percentile du revenu parental et la seconde le pourcentage de personnes âgées de 19 à 21 ans inscrite dans un établissement d'enseignement supérieur. Utiliser la première partie de l'exercice pour représenter ces données. Que pouvez-vous dire ?

   On pourra notamment ouvrir le fichier `csv` et stocker son contenu dans la liste `Liste` grâce aux lignes suivantes

```
with open("PercenRevParlVSEtude.csv", newline='') as csvfile:
    data = csv.reader(csvfile, delimiter=',')
    Liste=[]
    for row in data:
        Liste.append(list(row))
```

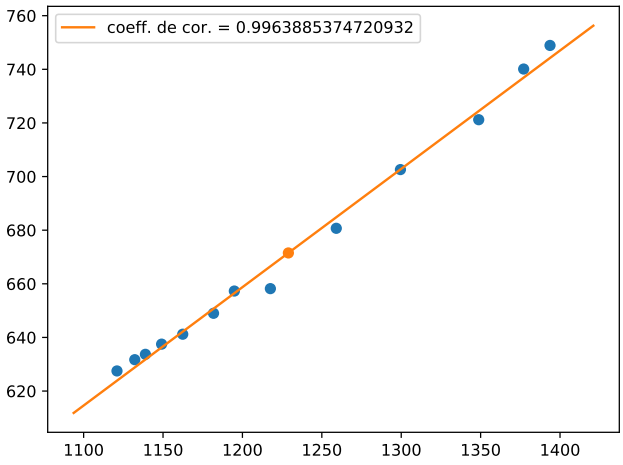
3. *Générations d'exemples types :* La fin de l'exercice est pour illustrer le fait qu'il faut considérer le coefficient de corrélation avec précaution.

a) On considère les données présentes dans les fichiers `ExAsetA.csv` et `ExAsetB.csv`. Afficher les représentations de ces deux jeux de données ainsi que les coefficients de corrélation r_1 et r_B des ensembles A et B ainsi que le coefficient de corrélation de l'ensemble $A \cup B$.

b) On considère les points dont les coordonnées sont stockées dans le fichier `ExBsetA.csv`. Représenter les données et faire apparaître le coefficient de corrélation. On considère un nouveau point dont les coordonnées sont contenues dans le fichier `ExBsetB.csv`. Représenter l'ensemble $A \cup B$ et faire apparaître le coefficient de corrélation.

c) Sur ces deux dernier exemples, on va illustrer le fait que le coefficient de correlation mesure un lien linéaire entre deux caractères. Représenter les données des fichiers `ExCsetA.csv` et `ExCsetB.csv`.

FIGURE 2 – Graphe de contrôle



EXERCICE 2.2 – DES DONNÉES "TROMPEUSES" – On veut étudier la série statistique à deux variables stockée dans le fichier suivant :

`regLinData.dat`

que l'on pourra importer de la manière suivante

```
import numpy as np
from scipy.stats import linregress
x, y = np.loadtxt("regLinData.dat", unpack=True)
```

1. a) Après avoir importé les données de la série statistique, tracer le nuage de points correspondant à ces données ainsi que la droite de régression linéaire correspondante en utilisant la fonction `linregress` de la bibliothèque `scipy.stats`.

   b) Tracer le nuage de points des points $(x_i, y_i - (ax_i + b))$ où les réels a et b sont les coefficients de la droite de régression. Les nombres $y_i - (ax_i + b)$ sont appelés *résidus*.

On constate alors que les résidus ne sont pas répartis autour de 0 et qu'ils présentent une structure parabolique.

2. a) Utiliser la fonction `np.polyfit` pour trouver les coefficients d'une fonction polynomiale du second degré permettant une meilleure description du nuage de points.

   b) Faire apparaître le nuage de points des résidus correspondant. Constater graphiquement que cette régression polynomiale est meilleure que la régression linéaire.

TP 3 – Utilisation de fréquences : analyse de messages chiffrés

Exercice 3.1 – Un calcul de fréquence – On se propose de faire de "une analyse fréquentielle" de la langue française, puis de se servir de celle-ci pour analyser quelques messages chiffrés.

En utilisant les ouvrages classiques fournis, construisez la liste des fréquences (une approximation de ces fréquences) d'apparition des lettres en français. On identifiera les caractères accentués au même caractère non accentué : par exemple les caractères 'é', 'è' et 'ê' seront comptés comme des 'e'.

On sauvegardera les fréquences obtenues dans un fichier `freq_lettre_fr_approx.json`. Pour sauvegarder un dictionnaire en un `.json`, on pourra utiliser les commandes suivantes.

```
# On importe la librairie json.
import json
# On ouvre un fichier.
new_file = open("nom_fichier.json","w")
# On sauvegarde le dictionnaire dictionary dans le fichier.
json.dump(dictionary,new_file)
# On ferme le fichier.
new_file.close()
```

Finissons cet exercice par une remarque : la fréquence des lettres en français dépend du style littéraire (par exemple, dans un roman, les pronoms "tu", "nous", "vous" apparaissent beaucoup plus que dans un texte scientifique, ce qui peut avoir une importance sur la fréquence d'apparition de la lettre "u"), dépend de l'auteur, etc.

Exercice 3.2 – Chiffrement de César – Le chiffrement de César est une méthode de chiffrement d'un texte en remplaçant chaque lettre du message par un décalage de $p \in [\![1, 25[\![$ de celle-ci. Par exemple, on peut appliquer utiliser le décalage suivant :

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

(On a effectué ici une permutation de l'alphabet). Par exemple, le message MICHEL devient une fois crypté le mot PLFKHO. Dans la suite, on aura régulièrement besoin de l'alphabet, que l'on peut générer ainsi :

```
#On importe la librairie string
import string
#on génère l'alphabet en minuscule
alphabet = string.ascii_lowercase
#on génère l'alphabet en majuscule
ALPHABET = string.ascii_uppercase
```

1. Écrire une fonction `caesar_crypt(msg,decal)` qui prend en entrée une chaîne de caractère `msg` et un nombre `decal` et qui renvoie le message crypté à la manière de César.

Dans la suite de cet exercice, on va montrer que ce type de chiffrement est très mauvais et ne résiste pas à une petite analyse statistique. Pour la suite, vous pourrez à loisir utiliser le fichier `freq_lettre_fr.json` qui contient la fréquence d'apparition des lettres en français. Pour récupérer le contenu d'un fichier `file.json` dans un dictionnaire, on pourra utiliser les lignes suivantes.

```
#On importe la librairie json
import json
#On ouvre le fichier file.json
fichier = open("file.json","r")
#On stocke les informations dans un dictionnaire.
dictionnaire = json.load(fichier)
#On ferme le fichier
fichier.close()
```

2. a) Écrire une fonction `bar_freq(fichier)` qui prend en entrée un nom de fichier `.json` contenant des fréquences de lettres en français et qui affiche le diagramme en barres de ces fréquences.

   b) Écrire une fonction `bar_freq1_freq2(fichier1,fichier2)` qui prend en entrée deux noms de fichier `.json` contenant deux séries de fréquences de lettres et les affichent sur le même diagramme.

   c) Écrire une fonction `bar_compare_freq(fichier1,fichier2,decal)` qui prend en entrée deux noms de fichier `.json` contenant deux séries de fréquences de lettres et les affichent sur le même diagramme en ayant fait subir un décalage de `decal` à la seconde série.

   d) À l'aide de cette dernière fonction, essayez de décrypter le texte crypté contenu dans le fichier `encrypted_with_caesar.txt`

3. Vous avez réussi à déterminer *graphiquement* quel était l'entier $k \in [\![1, 25]\!]$ utilisé pour chiffrer le texte précédent avec le chiffrement de César. On va essayer d'automatiser cette recherche.

   a) Écrire une fonction `correlation_freqs(fichier)` qui prend en entrée un nom de fichier contenant les fréquences d'apparition des lettres dans un texte chiffré avec César et qui calcule, pour chaque entier $n \in [\![0, 25]\!]$, le coefficient de corrélation entre les fréquences des lettres en français et celles contenues dans `fichier` décalées de n. La fonction renvoie le graphe de ces coefficients de correlations en fonction de n.

   b) En modifiant la fonction précédente, écrire une fonction `decrypt(fichier)` qui prend en entrée un fichier avec un texte crypté et qui crée un fichier contenant le message de `fichier` déchiffré.

Une dernière remarque : nous avons montré sur un exemple que le chiffrement de César est très simple à casser sous une condition. En effet, il faut connaître la langue du message initial.

Exercice 3.3 – Chiffrement par substitution – On présente ici une autre méthode de chiffrement : le *chiffrement par substitution*. Le but de l'exercice est de produit un programme permettant de déchiffrer automatiquement un texte chiffré avec cette méthode. La stratégie employé ici est une de celles présentées dans l'article suivant :

> Didier Müller, *Les métaheuristiques en cryptanalyse*, Bulletin de la SSPMP, numéro 143. Mai 2020.

Le chiffrement par substitution fonctionne comme suit. On considère un message textuel sans accent sans ponctuation et sans espace. On choisit une permutation σ de l'alphabet (une application bijective de l'alphabet dans lui-même). Par exemple, on pourrait donner la permutation suivante :

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
K	F	N	H	X	U	B	C	D	T	S	O	G	R	J	P	Y	M	L	E	A	I	V	W	Z	Q

Utilisant cette permutation σ, on remplace chaque lettre du message par son image par σ. Dans la suite, une telle permutation de l'alphabet sera appelé *clé*. Pour retenir une clé, on ne retiendra la chaîne des caractères de l'alphabet permuté. Par exemple, on pourra considérer la clé suivante :

```
key = "KFNHXUBCDTSOGRJPYMLEAIVWZQ"
```

1. Écrire une fonction `crypt(msg)` qui prend en entrée un message textuel `msg`, qui supprime toute ponctuation, espace ou accents, mettant toutes les lettres en majuscule et qui renvoie le message en l'ayant crypté grâce à un choix aléatoire de permutation des lettres. Pour construire cette fonction, vous pourrez notamment utiliser la fonction suivante :

```python
#On importe la librairie random
import random
#On teste ce que fait la fonction suivante
def string_shuffle(word):
    """
    word est une chaine de caractères
    """
    c = list(word)
    random.shuffle(word)
    return ''.join(c)
```

2. Combien y-a-t-il de chiffrements possibles avec cette méthode ? Est-il raisonnable de vouloir déchiffrer ce message en essayant tous les chiffrements possibles ?
3. En utilisant le code de la précédente fonction, écrire une fonction `chiffre(msg,key)` qui prend en entrée une chaîne de caractères `msg` et une clé `key` et qui renvoie la chaîne de caractère chiffré grâce à la clé.

Vous avez récupéré un texte en français qui a été chiffré par substitution. Celui-ci est accessible dans le fichier

`texte_chiffre_substitution.txt`

Le but de la fin de cet exercice est de décoder ce message.

4. On commence par étudier les fréquences de chacune des lettres du texte.
   a) Calculer les fréquences d'apparition de chaque lettre.
   b) À la manière de l'exercice 3.2, essayez de décoder le texte.

Normalement, cette tentative échoue (ou alors vous avez beaucoup de chance) car les fréquences des lettres dans le texte crypté sont trop proches pour pouvoir conclure.

Afin de pouvoir décoder un texte de cette manière, l'idée est de ne plus étudier les fréquences des lettres mais d'étudier les fréquences des groupes de lettres. On pourrait essayer avec les digrammes (des couples de lettres comme EN, NT, IS, ...) Mais ce n'est pas très efficace. Dans la suite, nous présentons une méthode qui fait une attaque fréquentielle du chiffrement en utilisant des fréquences de quadrigrammes en français (des blocs de quatre lettres MENT, USSI, MALA, ...). Pour cela, on fournit le fichier

`quadrigrammes.txt`

qui contient sur chaque ligne un quadrigramme et le nombre d'apparition de ce quadrigramme dans un corpus de textes représentatif de la langue française.

5. Écrire une fonction qui lit le fichier `quadrigrammes.txt` et qui renvoie un dictionnaire `{"quad": N}` qui possède chaque quadrigramme en clé et le nombre d'occurrences. Faites en sorte que cette fonction sauvegarde ce dictionnaire dans un fichier `.json`. On retiendra ce dictionnaire dans une variable globale `DICT_QUAD`.

À un texte chiffré, on attribue un score, appelé *logscore* grâce à la fonction suivante :

```python
import numpy as np
def logscore(chain):
    """
    Calcule le log_score de la chaîne de caractère chain
    grace au dictionnaire de quadrigrammes
    """
    logsum = 0
    min_freq = 1e-10
    for i in range(len(string)-3):
        logsum += np.log10(DICT_QUAD.get(chain[i:i+4], min_freq))
    return -logsum
```

Ainsi, plus un texte est proche du français, plus le logscore renvoyé par cette fonction sera faible. Pour la fin de cet exercice, nous allons coder une série de fonction afin d'automatiser le décodage des textes chiffrés par substitution. On suit l'algorithme suivant

6. On commence par générer une clef `key` aléatoire avec laquelle on déchiffre le texte et on calcule son logscore ;
7. On considère toutes les clefs voisines `key`. Une clé `key2` est voisine de la clé `key` si
   — on peut passer de `key` à `key2` en permutant deux caractères ; par exemple les clés

`"KFNHXUBCDTSOGRJPYMLEAIVWZQ"` et `"KFNHXQBCDTSOGRJPYMLEAIVWZU"`

sont voisines car on a échangé le `U` et le `Q`;

— on peut passer de `key` à `key2` en déplaçant un unique caractère dans la chaîne; par exemple les clés

`"KFNHXUBCDTSOGRJPYMLEAIVWZQ"` et `"KFNHXUBDTSOGRJPYMLEAIVWZQC"`

sont voisines car on a déplacé le `C` .

De ces clés voisines, on ne conserve que celle qui a le logscore le plus bas et on retient les autres dans une liste `OldKey`.

8. On réitère le procédé avec la nouvelle clé en évitant de tester les clés présentes dans la liste `OldKey`.
9. On s'arrête lorsque l'algorithme a tourné un certain nombre de fois ou alors que la clé retenue à un logscore plus bas que toutes ces voisines.

TP 4 – Visualisation de données géographiques (Sae)

Exercice 4.1 – Représentation de données spatiales – Un certain nombre de données sont des indicateurs statistiques croisés avec une donnée géographique. Pour tenir compte de cette donnée géographique, nous pouvons alors représenter nos données statistiques à l'aide d'une carte. C'est l'objectif de cet exercice. On utilisera les fichiers :

`a-dep2021.json` et `collegiens_REP.csv`

qui proviennent respectivement de

— https://www.data.gouv.fr/fr/datasets/contours-des-communes-de-france-simplifie-avec-regions-et-departement-doutre-mer-rapproches/

— https://www.insee.fr/fr/statistiques/4797582?sommaire=4928952&q=Colleges+REP%2B#tableau-figure3

1. On commence par étudier le fichier `a-dep2021.json`. C'est un fichier `.json` qui contient notamment des données géographique. Afin de l'ouvrir, on va utiliser la librairie `geopandas`, de la manière suivante.

```
import geopandas as gpd
### Lecture du fichier
data = gpd.read_file("a-dep2021.json")
```

2. Afficher le contenu de `data` en exécutant la commande `print(data)`. La variable `data` est du type `panda.dataframe` qui contient une colonne `geometry` qui contient des données géographiques.
3. Affichons une représentation des données géographiques grâce aux commandes suivantes :

```
import pandas as pd
import geopandas as gpd
### Lecture du fichier
data = gpd.read_file('a-dep2021.json')
### Ouverture d'une figure
fig, axes = plt.subplots(1,1,figsize=(12,12))
### Tracé des données géographiques dans la figure
data.plot(
    ax = axes, # Axes de tracé
    linewidth=0.1, # Epaisseur de la ligne
    edgecolor='black', # Couleur de la ligne
    color='blue', # Couleur de l'aplat
    alpha = 0.5 # Transparence de l'aplat
)
axes.set_axis_off() # Suppression des axes
### Affichage
plt.show()
```

4. On a affiché les formes des différents départements français. Il pourrait être intéressant de les contextualiser ; pour cela on va chercher à inclure cela sur une "vraie varte". *Openstreetmap* fournit son lot de cartes libres d'accès. Pour cela :
   a) on importe la librairie `contextily` avec l'ajout d'une ligne `import contextily as ctx` ;
   b) avant toute opération de tracé, on rattache nos données à cette carte avec l'ajout d'une ligne `data = data.to_crs(epsg=3857)` ;
   c) on affiche la carte avec l'ajout d'une ligne `ctx.add_basemap(ax=axes)` .

   Vous devriez constater que les DOM apparaissent en Espagne, ce qui n'est pas l'idéal.
5. On va filtrer nos données afin de ne considérer que la métropole. Pour cela, ajouter les lignes suivantes dans votre code. Ces lignes permettent de ne conserver que les lignes dont l'élément de la colonne `"dep"` n'est pas un élément de `filter_dep` .

```
### Code départementaux des DOM
filter_dep = ["971", "972", "973", "974", "976"]
### On filtre nos données
data = data.query('dep not in @filter_dep')
```

6. On voudrait à présent que la couleur de chaque département corresponde à une donnée statistique ; on va chercher à illustrer le pourcentage de collégiens présent dans un établissement REP+ dans chaque département. En utilisant ce qu'y a été fait précédemment, faire une fonction qui permet d'afficher la carte présente en Figure 3.

Pour cela, vous pourrez utiliser :

— la liste de de couleurs

```
colors=['#FFFFFF','#E2B5A4','#C48B7D','#A76258','#8A3832','#6C0000']
```

— vous pourrez ajouter une colonne à `data` à partir d'une liste de la bonne taille

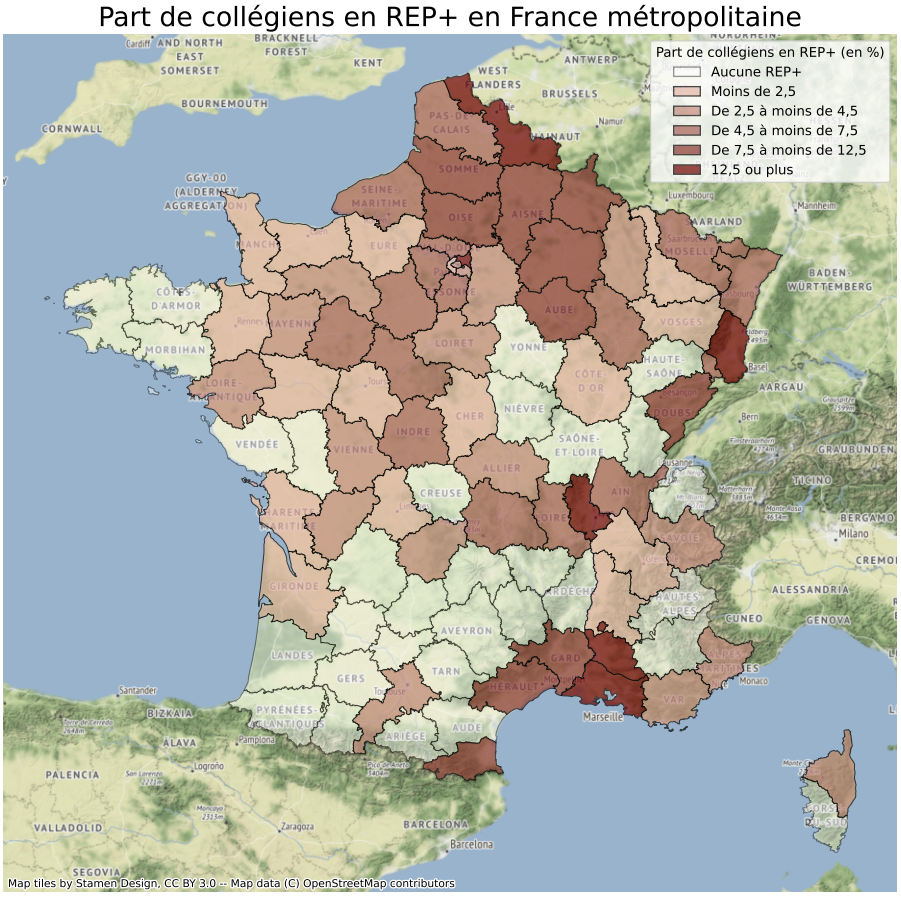
```
data['new_column'] = pd.DataFrame(liste_valeurs)
```

— vous pouvez filtrer `data` grâce à une condition. Par exemple `data[data['dep'] <= 44]` qui retourne les données pour tous les départements ayant un indicatif inférieur ou égal à 44 ;

— la légende est personnalisable grâce à la fonction `Patch` de la librairie `matplotlib.patches` et l'argument optionnel `handles` de la fonction `axes.legend` (où `axes` est le nom des axes de la figure).

Exercice 4.2 – Représentation de données géographiques et temporelles – évolution du Covid en France sous forme de Gif – Cet exercice est dans la directe continuité de l'exercice précédent. Le but est de créer un gif montrant l'évolution du nombre d'hospitalisations dues au Covid19. On utilisera les fichiers de données suivants :

FIGURE 3 – Carte générée par python : collégiens en REP+ en France métropolitaine



`a-dep2021.json`, `data_covid_hosp.csv` et `nb_hab_par_dep.csv`.

Les données relatives aux cas de Covid19 proviennent de

`https://www.data.gouv.fr/fr/datasets/donnees-hospitalieres-relatives-a-lepidemie-de-covid-19/`

alors que les données relatives au nombre d'habitants par département proviennent de

`https://www.insee.fr/fr/statistiques/1893198`.

Pour tout l'exercice, n'hésitez pas à consulter toute la documentation nécessaire sur Internet.

1. Faire une fonction `mise_en_forme_data()` qui
   - lit les deux fichiers `data_covid_hosp.csv` et `a-dep2021.json` ;
   - qui récupère dans `data_covid_hosp.csv` les hospitalisations liées au Covid et qui, pour chaque date, l'ajoute comme une nouvelle colonne aux données de `a-dep2021.json` avec la date comme nom de colonne ;

— qui filtrer les données pour pour ne garder que les données de la métropole ;

— qui sauvegarde ces données remises en forme dans un fichier `covid_dep_abs.geojson`.

Pour cela, vous pourrez notamment utiliser les commandes introduites dans l'exercice précédent, mais également

— la méthode `merge` pour fusionner les dataframes ;

— la méthode `rename` pour renommer des colonnes d'une dataframe ;

Maintenant que nos données sont toutes stockées dans le même fichier, nous allons pouvoir créer les images qui serviront à la création de notre gif.

2. Commencer par faire une fonction `visualisation_data(date)` où `date` est une date en chaine de caractères apparaissant comme nom de colonne dans les données de `covid_dep_abs.geojson`. Cette fonction lit `covid_dep_abs.geojson` et affiche la carte de la métropole avec les départements colorés en fonction du nombre de cas d'hospitalisations dues au Covid19. Vous pourrez notamment utiliser les arguments optionnels `column` et `cmap` de la méthode `plot` qui contrôle respectivement le nom de la colonne servant à colorer la carte et la manière de colorer.

```
fig, axes = plt.subplots(1,1,figsize=(10,10))
data.plot(ax=axes, column=date, cmap='magma_r',legend=True)
```

Attention : dans la suite, le temps d'execution des fonctions peut-être relativement long.

3. En reprenant la fonction précédente, faire une fonction `save_maps()` qui créer un dossier `temp` et qui sauvegarde dedans les cartes produites à partir de chacune des dates des données d'hospitalisation. Vous pourrez notamment utiliser

— la librairie `os` afin de créer un dossier avec `os.makedir` ;

— les arguments optionnels `vmin` et `vmax` de la méthode `plot` des dataframes afin la même taille de légende pour toutes les images.

Nous pouvons nous atteler à la création du gif. Pour cela, nous allons importer la fonction `Image` de la librairie `PIL`.

4. Modifier la fonction précédente de la manière suivante :

— créer une liste vide `frames=[]` avant la génération des images ;

— à chaque création d'image faire `frames.append(Image.open(name))` où `name` est le nom de l'image ;

— enfin on sauvegardera notre gif grâce à la commande suivante, puis on supprimera le dossier `temp` grâce à la fonction `rmtree` de la librairie `shutil` :

```
frames[0].save(
    "covid_hosp_abs.gif", # nom du fichier sauv
    format="gif", # Format
    append_images=frames[1:],
    save_all=True,
    duration=100, # Temps entre 2 frames
    loop=0, # Nombre de boucles
)
```

Nous avons donc un gif qui nous montre l'évolution au cours du temps du nombre de personnes hospitalisées à cause du Covid19. Vous devez constater que lors des pics de l'épidémie, la région Nord et la région Île de France apparaissent comme très touchées contrairement à la grande majorité du territoire ; cela ne pourrait être dû qu'au fait que ces régions sont beaucoup plus peuplées que d'autres (et qu'il est donc normal qu'il y ait plus de cas.)

5. En utilisant les données présentes dans le fichier `nb_hab_par_dep.csv`, reprenez la fonction précédente pour générer un gif affichant la proportion d'habitants hospitalisés pour cause du Covid19 par département.

Email address: johan.leray@univ-nantes.fr

Département d'informatique – IUT de Nantes