Récap notebook TER

Récap fonctions

▼ Code def fonction

```
def string_found(kw, txt):
    """
        Parameters
        ----------
        kw : str
            The keyword to search
        txt : str
            The string to search into

        Returns
        -------
        str
            The matched keywords and number of occurrences
    """
    sg = re.findall(r"\b" + re.escape(kw) + r"\b", txt)
    pl = re.findall(r"\b" + re.escape(kw) + "s"+ r"\b", txt)
    if kw[-1] == 's':
        kws = kw[:-1]
        sg2 = re.findall(r"\b" + re.escape(kws) + r"\b", txt)
    else:
        sg2 = []
    occ = sg + pl + sg2
    if len(occ) >= 1:
        return('{} ({})'.format(kw, len(occ)))
    else:
        return("")

def list_string_found(kw, txt):
    """
        Parameters
        ----------
        kw : list
            A list of keywords to search
        txt : str
            The string to search into

        Returns
        -------
        str
            The matched keywords and number of occurrences
    """
    f = ""
    for k in kw:
        matches = string_found(k, txt)
        if len(matches) > 0:
            if len(f) > 0 :
                f = f + " / {}".format(matches)
            else:
                f = matches
    return(f)

def df_freq(l, colnames = ["fr", "freq"]):
    """
        Parameters
        ----------
        l : list
            List of sets (keyword, freq)

        colnames : list, optional
```

```
            The names of the columns in the output df

        Returns
        -------
        df
            Dataframe of the matched keywords and their frequency
    """
    nb = sum([x[1]for x in l])
    kw = [x[0]for x in l]
    freq = [x[1]*100/nb for x in l]
    df = pd.DataFrame({"kw" : kw, "freq" : freq})
    df.columns = colnames
    return(df)

def freq_lexicon(lex, txt):
    """
        Parameters
        ----------
        lex : list
            Lexicon
        txt : str
            The string to search into

        Returns
        -------
        dict
            The matched keywords and number of occurrences
    """
    string_found = [list_string_found(lex, x) for x in txt]
    string_found = " / ".join(string_found)
    w = [k.split(" (")[0] for k in string_found.split(" / ") if len(k) >1]
    f = [int(k.split(" (")[1].replace(")", "")) for k in string_found.split(" / ") if len(k) >1]
    unique_w = list(set(w))
    dic_freq, dic_freq_doc = {}, {}
    for word in unique_w:
        idx = [i for i, j in enumerate(w) if j == word]
        freq = sum([f[i] for i in idx])
        freq_doc = w.count(word)
        dic_freq_doc[word] = freq_doc
        dic_freq[word] = freq
    ordered_dic = sorted(dic_freq.items(), key=lambda x:x[1], reverse=True)
    ordered_dic_doc = sorted(dic_freq_doc.items(), key=lambda x:x[1], reverse=True)
    return({'occurrences': ordered_dic, 'doc': ordered_dic_doc})
```

1. `string_found(kw, txt)`: Cette fonction prend en entrée un mot-clé (`kw`) et une chaîne de texte (`txt`) et recherche les occurrences du mot-clé dans le texte. Elle compte les occurrences simples, les occurrences au pluriel (si le mot-clé se termine par 's') et les occurrences de la forme singulière du mot-clé (si applicable). Elle renvoie une chaîne de caractères contenant le mot-clé et le nombre total d'occurrences.
2. `list_string_found(kw, txt)`: Cette fonction prend en entrée une liste de mots clés (`kw`) et une chaîne de texte (`txt`). Elle itère à travers la liste de mots clés et utilise la fonction `string_found` pour trouver des correspondances pour chaque mot clé dans le texte. Elle renvoie une chaîne de caractères contenant les mots clés correspondants et leur nombre respectif.
3. `df_freq(l, colnames=["fr", "freq"])`: Cette fonction prend en entrée une liste de paires de mots-clés et de fréquences (`l`) ainsi que des noms de colonnes (par défaut "fr" et "freq"). Elle calcule la fréquence totale et les pourcentages de fréquence pour chaque mot-clé et renvoie les résultats sous forme d'un DataFrame Pandas.
4. `freq_lexicon(lex, txt)`: Cette fonction prend en entrée un lexique (une liste de mots clés) et une chaîne de texte (`txt`). Elle utilise la fonction `list_string_found` pour trouver des correspondances pour chaque mot clé dans le texte, puis calcule le nombre total d'occurrences de chaque mot clé dans le texte et à travers les documents. Elle retourne un dictionnaire contenant les occurrences des mots clés triées par fréquence et fréquence documentaire.

Ce script est conçu pour analyser la fréquence de mots-clés spécifiques dans un texte donné ou un ensemble de textes, et générer des statistiques de fréquence et des classements.

1. The lexicons

▼ Code lexicons

```
expert_lexicon = pd.read_excel('D:/Mes Donnees/Data/Lexiques/expert.xlsx')
expert_lexicon_conv = pd.read_excel('D:/Mes Donnees/Data/Lexiques/expert.xlsx',
sheet_name = "category")
expert_G = list(set([w for w in expert_lexicon['LEXG'].tolist() if str(w) != 'nan']))
expert_A = list(set([w for w in expert_lexicon['LEXA'].tolist() if str(w) != 'nan']))
expert_C = list(set([w for w in expert_lexicon['LEXC'].tolist() if str(w) != 'nan']))

len(list(set(expert_G+expert_A+expert_C)))
```

1. `expert_lexicon = pd.read_excel('D:/Mes Donnees/Data/Lexiques/expert.xlsx')`: Cette ligne lit le contenu d'un fichier Excel situé dans le chemin de fichier spécifié ('D:/Mes Donnees/Data/Lexiques/expert.xlsx') dans un DataFrame Pandas nommé `expert_lexicon`. Par défaut, cela lit la première feuille du fichier Excel.
2. `expert_lexicon_conv = pd.read_excel('D:/Mes Donnees/Data/Lexiques/expert.xlsx', sheet_name="category")`: Cette ligne lit une feuille spécifique nommée "category" du même fichier Excel et stocke son contenu dans un autre DataFrame Pandas nommé `expert_lexicon_conv`.
3. `expert_G`, `expert_A` et `expert_C` Les listes sont créées en extrayant et en traitant les données des colonnes 'LEXG', 'LEXA' et 'LEXC' du DataFrame `expert_lexicon`. Ces listes sont filtrées pour supprimer les valeurs 'nan' et représentent les valeurs distinctes de ces colonnes.
4. Enfin, `len(list(set(expert_G+expert_A+expert_C)))` calcule le nombre total de valeurs distinctes dans les trois listes ('expert_G', 'expert_A' et 'expert_C') en les combinant en une seule liste, en la convertissant en ensemble pour supprimer les doublons, puis en la reconvertissant en liste. La fonction `len` est utilisée pour compter les éléments de cette liste, vous donnant ainsi le nombre total de valeurs distinctes.

Cela peut être utile pour déterminer le nombre total de termes ou d'éléments uniques dans ces listes, ce qui peut être pertinent pour diverses tâches d'analyse ou de traitement de données. L'exécution de ce code donne le compte des valeurs uniques dans ces listes.

KEOPS lexicon (FR)

▼ Code KEOPS FR

```
keops_lexicon = pd.read_excel('D:/Mes Donnees/Data/Lexiques/KEOPS.xlsx', sheet_name = "Food_security_bilingual")
keops_lexicon_fr = list(set(keops_lexicon['fr']))
keops_lexicon_fr = [x.lower().strip() for x in keops_lexicon_fr]
keops_lexicon_en = list(set(keops_lexicon['en']))
keops_lexicon_en = [x.lower().strip() for x in keops_lexicon_en]
len(keops_lexicon_fr)
```

1. `keops_lexicon = pd.read_excel('D:/Mes Donnees/Data/Lexiques/KEOPS.xlsx', sheet_name="Food_security_bilingual")`: Cette ligne lit le contenu de la feuille spécifiée ("Food_security_bilingual") du fichier Excel 'KEOPS.xlsx' dans un DataFrame Pandas nommé `keops_lexicon`.
2. `keops_lexicon_fr = list(set(keops_lexicon['fr']))`: Cette ligne extrait les valeurs uniques de la colonne 'fr' du DataFrame `keops_lexicon` et les stocke dans une liste nommée `keops_lexicon_fr`.

3. `[x.lower().strip() for x in keops_lexicon_fr]` : Cette compréhension de liste itère sur chaque élément de la liste `keops_lexicon_fr`, les convertit en minuscules en utilisant `lower()`, et supprime les espaces vides au début et à la fin en utilisant `strip()`. Cela garantit que tous les termes français sont en minuscules et n'ont pas d'espaces vides au début ou à la fin.
4. `keops_lexicon_en = list(set(keops_lexicon['en']))` : De manière similaire, cette ligne extrait les valeurs uniques de la colonne 'en' du DataFrame `keops_lexicon` et les stocke dans une liste nommée `keops_lexicon_en`.
5. `len(keops_lexicon_fr)` : Enfin, cette ligne calcule la longueur de la liste `keops_lexicon_fr`, ce qui représente le nombre de termes français uniques dans la colonne 'fr' de la feuille Excel.

Ainsi, l'expression `len(keops_lexicon_fr)` donne le nombre de termes français uniques dans la colonne 'fr' de la feuille Excel 'KEOPS.xlsx' après les avoir convertis en minuscules et avoir supprimé les espaces vides au début et à la fin.

Lexique Fraiberger (EN)

▼ Code Fraiberger EN

```
fraibeger_lexicon = pd.read_excel('D:/Mes Donnees/Data/Lexiques/fraiberger.xlsx', sheet_name='bilingual')
fraibeger_lexicon['en'] = fraibeger_lexicon['en'].apply(lambda x:x.lower().strip())
fraibeger_lexicon['fr'] = fraibeger_lexicon['fr'].apply(lambda x:x.lower().strip())
fraibeger_lexicon = fraibeger_lexicon.drop_duplicates()
fraibeger_lexicon_en = list(set(fraibeger_lexicon['en']))
fraibeger_lexicon_fr = list(set(fraibeger_lexicon['fr']))

len(fraibeger_lexicon_fr)
```

1. `fraibeger_lexicon = pd.read_excel('D:/Mes Donnees/Data/Lexiques/fraiberger.xlsx', sheet_name='bilingual')` : Cette ligne lit le contenu de la feuille spécifiée ('bilingual') du fichier Excel 'fraiberger.xlsx' dans un DataFrame Pandas nommé `fraibeger_lexicon`.
2. `fraibeger_lexicon['en'] = fraibeger_lexicon['en'].apply(lambda x: x.lower().strip())` : Cette ligne applique une fonction lambda à la colonne 'en' du DataFrame `fraibeger_lexicon`, convertissant chaque valeur en minuscules avec `lower()` et supprimant les espaces vides au début et à la fin avec `strip()`.
3. `fraibeger_lexicon['fr'] = fraibeger_lexicon['fr'].apply(lambda x: x.lower().strip())` : De manière similaire, cette ligne applique la même fonction lambda à la colonne 'fr' du DataFrame `fraibeger_lexicon` pour garantir que les termes français sont en minuscules et n'ont pas d'espaces vides au début ou à la fin.
4. `fraibeger_lexicon = fraibeger_lexicon.drop_duplicates()` : Cette ligne supprime les lignes en double du DataFrame `fraibeger_lexicon`, ne conservant que les lignes uniques.
5. `fraibeger_lexicon_en = list(set(fraibeger_lexicon['en']))` : Cette ligne extrait les valeurs uniques de la colonne 'en' du DataFrame `fraibeger_lexicon` nettoyé et les stocke dans une liste nommée `fraibeger_lexicon_en`.
6. `fraibeger_lexicon_fr = list(set(fraibeger_lexicon['fr']))` : De manière similaire, cette ligne extrait les valeurs uniques de la colonne 'fr' du DataFrame `fraibeger_lexicon` nettoyé et les stocke dans une liste nommée `fraibeger_lexicon_fr`.
7. `len(fraibeger_lexicon_fr)` : Enfin, cette ligne calcule la longueur de la liste `fraibeger_lexicon_fr`, ce qui représente le nombre de termes français uniques dans la colonne 'fr' de la feuille Excel après le nettoyage et la suppression des doublons.

Ainsi, l'expression `len(fraibeger_lexicon_fr)` donne le nombre de termes français uniques dans la colonne 'fr' de la feuille Excel 'fraiberger.xlsx' après les avoir convertis en minuscules, avoir supprimé les espaces vides

au début et à la fin, et avoir supprimé les doublons.

All lexicons

```
all_lexicons = list(set(keops_lexicon_fr + list(set(expert_G+expert_A+expert_C)) + fraibeger_lexicon_fr))
len(all_lexicons)
```

Ce code combine les valeurs uniques de trois listes différentes (`keops_lexicon_fr`, `expert_G+expert_A+expert_C` et `fraibeger_lexicon_fr`) en une seule liste appelée `all_lexicons`. Il calcule ensuite la longueur de cette liste combinée pour déterminer le nombre total d'éléments uniques dans les trois listes.

1. `keops_lexicon_fr + list(set(expert_G+expert_A+expert_C)) + fraibeger_lexicon_fr` : Cette partie combine les termes français uniques de la liste `keops_lexicon_fr`, les termes uniques combinés des listes `expert_G`, `expert_A` et `expert_C`, et les termes français uniques de la liste `fraibeger_lexicon_fr` en une seule liste.
2. `list(set(...))` : Cela est utilisé pour s'assurer que la liste résultante ne contient que des éléments uniques en la convertissant en ensemble (set) puis en liste.
3. `len(all_lexicons)` : Enfin, cette ligne calcule la longueur de la liste `all_lexicons`, qui représente le nombre total d'éléments uniques dans les trois listes.

Donc, `len(all_lexicons)` donne le nombre d'éléments uniques lorsqu'on combine les termes français uniques des listes `keops_lexicon_fr`, `expert_G+expert_A+expert_C` et `fraibeger_lexicon_fr`.

2. Coverage between lexicons

▼ Code coverage

```
x1 = len(keops_lexicon_fr)
x2 = len(set(expert_G + expert_A + expert_C))
x3 = len(set(expert_G + expert_A + expert_C).intersection(keops_lexicon_fr))
x4 = len(set(expert_G ))
x5 = len(set(expert_G).intersection(keops_lexicon_fr))
x6 = len(set(expert_A ))
x7 = len(set(expert_A).intersection(keops_lexicon_fr))
x8 = len(set(expert_C ))
x9 = len(set(expert_C).intersection(keops_lexicon_fr))
x10 = len(keops_lexicon_en)
x11 = len(fraibeger_lexicon_en)
x12 = len(set(keops_lexicon_en).intersection(fraibeger_lexicon_en))
x13 = len(fraibeger_lexicon_fr)
x14 = len(set(expert_G + expert_A + expert_C).intersection(fraibeger_lexicon_fr))
```

Ce code calcule divers décomptes et intersections entre différents ensembles de données.

- `x1` : Nombre de termes français uniques dans `keops_lexicon_fr`.
- `x2` : Nombre de termes uniques dans `expert_G + expert_A + expert_C` (lexique combiné).
- `x3` : Nombre de termes uniques communs entre `keops_lexicon_fr` et le lexique combiné.
- `x4` : Nombre de termes uniques dans `expert_G` (le lexique de la colonne 'LEXG' de 'expert.xlsx').
- `x5` : Nombre de termes uniques communs entre `keops_lexicon_fr` et `expert_G`.
- `x6` : Nombre de termes uniques dans `expert_A` (le lexique de la colonne 'LEXA' de 'expert.xlsx').
- `x7` : Nombre de termes uniques communs entre `keops_lexicon_fr` et `expert_A`.

- `x8` : Nombre de termes uniques dans `expert_C` (le lexique de la colonne 'LEXC' de 'expert.xlsx').
- `x9` : Nombre de termes uniques communs entre `keops_lexicon_fr` et `expert_C`.
- `x10` : Nombre de termes anglais uniques dans `keops_lexicon_en`.
- `x11` : Nombre de termes anglais uniques dans `fraibeger_lexicon_en`.
- `x12` : Nombre de termes anglais uniques communs entre `keops_lexicon_en` et `fraibeger_lexicon_en`.
- `x13` : Nombre de termes français uniques dans `fraibeger_lexicon_fr`.
- `x14` : Nombre de termes français uniques communs entre le lexique combiné (`expert_G + expert_A + expert_C`) et `fraibeger_lexicon_fr`.

Ces variables représentent divers décomptes et intersections entre différents lexiques (en français et en anglais) et peuvent être utiles pour analyser le chevauchement et l'unicité des termes entre ces lexiques. Suivant les objectifs d'analyse, ces décomptes peuvent fournir des informations sur les relations entre les lexiques.

KEOPS vs Experts (french)

Vocabulary in KEOPS not in expert lexicons :

```
[x for x in keops_lexicon_fr if x not in set(expert_G + expert_A + expert_C)]
```

["système d'alerte rapide",
'aliment nouveau',
"accès à l'alimentation",
'sécurité alimentaire en milieu urbain',
'qualité nutritionnelle',
"systèmes d'alerte rapide",
'autosuffisance',
'sécurité alimentaire des ménages',
"droit à l'alimentation",
'nouveaux aliments',
'gestion des ressources',
'souveraineté alimentaire',
'sécurité alimentaire durable',
'durabilité socio-économique',
'intensification durable',
'insécurité alimentaire',
'sécurité nutritionnelle']

Vocabulary in KEOPS not in fraibeger_lexicon :

```
[x for x in keops_lexicon_en if x not in fraibeger_lexicon_en]
```

['malnutrition',
'sustainable intensification',
'hunger',
'right to food',
'self-sufficiency',

'novel foods',
'nutrition security',
'early warning',
'early warning systems',
'food aid',
'urban nutrition security',
'household food security',
'novel food',
'food sovereignty',
'resource management',
'socioeconomic sustainability',
'food access',
'nutritional quality',
'sustainable food security',
'food security']

3. Coverage between lexicons and corpus

```
data = [["FRESA corpus", 26178,"Burkina Faso, Bénin, Sénégal","2004-en cours","FR",17],
        ["BF corpus", 22856,"Burkina Faso","2009-2018","FR","2 (Burkina24, LeFaSo)"]]

pd.DataFrame(data, columns=["Corpus", "No articles", "Spatial coverage",
"Temporal coverage", "Language", "No of sources"]).style.hide(axis="index")
```

FRESA corpus

▼ Code FRESA

```
import glob
fresa_path = r'D:/Mes Donnees/Data/corpus_FRESA' # use your path
all_files = glob.glob(fresa_path + "/*.csv")

rows = []
for filename in all_files:
    df = pd.read_csv(filename, index_col=None, header=0,encoding='utf-8')
    rows.append(df)

FRESA = pd.concat(rows, axis=0, ignore_index=True).reset_index()

# Remove article without publishing date
FRESA = FRESA.dropna(subset=['published_at'])

# Removing duplicates
FRESA = FRESA.drop_duplicates(subset=['title', 'source'])

# Creating a column indicating if the source is Youtueb or an online news
FRESA['source_norm'] = ['youtube' if 'youtube' in i else 'online news' for i in FRESA.url]

# Lowercasing the text
FRESA['text_clean'] = FRESA['text'].apply(lambda x: x.lower())

# Creating time columns for aggregating
FRESA['year_month'] = FRESA['published_at'].apply(lambda x: str(x)[0:7])
FRESA['year'] = FRESA['published_at'].apply(lambda x: int(str(x)[0:4]))
FRESA['year_month'] = pd.to_datetime(FRESA['year_month'])


FRESA_agg = FRESA.groupby(['year_month', 'source_norm'])['index'].count().reset_index()
```

```
FRESA.head()
```

1. `glob.glob(fresa_path + "/*.csv")` : Cette ligne utilise le module `glob` pour récupérer une liste de chemins de fichiers pour tous les fichiers CSV dans le répertoire spécifié (`fresa_path`).
2. `rows = []` : Initialise une liste vide pour stocker les DataFrames individuels lus à partir de chaque fichier CSV.
3. Boucle à travers chaque fichier CSV:
   - `df = pd.read_csv(filename, index_col=None, header=0, encoding='utf-8')` : Lit chaque fichier CSV dans un DataFrame Pandas (`df`) en spécifiant la colonne d'index, la ligne d'en-tête et l'encodage des caractères.
   - `rows.append(df)` : Ajoute chaque DataFrame à la liste `rows` .
4. `FRESA = pd.concat(rows, axis=0, ignore_index=True).reset_index()` : Concatène tous les DataFrames de la liste `rows` le long de l'axe des lignes (`axis=0`), en ignorant les indices d'origine, puis réinitialise l'index du DataFrame résultant.
5. `FRESA = FRESA.dropna(subset=['published_at'])` : Supprime les lignes où la colonne 'published_at' est manquante (valeurs NaN).
6. `FRESA = FRESA.drop_duplicates(subset=['title', 'source'])` : Supprime les lignes en double basées sur les colonnes 'title' et 'source', en ne conservant que la première occurrence.
7. `FRESA['source_norm'] = ['youtube' if 'youtube' in i else 'online news' for i in FRESA.url]` : Crée une nouvelle colonne 'source_norm' qui catégorise les sources comme étant 'youtube' ou 'online news' en fonction de la présence du mot 'youtube' dans la colonne 'url'.
8. `FRESA['text_clean'] = FRESA['text'].apply(lambda x: x.lower())` : Crée une nouvelle colonne 'text_clean' contenant le texte en minuscules de la colonne 'text'.
9. `FRESA['year_month']` et `FRESA['year']` : Extrait les informations d'année et d'année-mois de la colonne 'published_at' et les convertit en objets datetime.
10. `FRESA_agg = FRESA.groupby(['year_month', 'source_norm'])['index'].count().reset_index()` : Effectue une agrégation en regroupant le DataFrame en fonction des colonnes 'year_month' et 'source_norm' et en comptant le nombre d'occurrences ('index') dans chaque groupe. Le résultat est stocké dans le DataFrame 'FRESA_agg'.

L'exécution de ce code, renvoie le DataFrame traité ('FRESA') contenant des données à partir des fichiers CSV, avec diverses transformations appliquées, ainsi qu'un DataFrame agrégé ('FRESA_agg') qui résume les données par année-mois et type de source.

Burkina Faso corpus

```
filename = r'D:/Mes Donnees/Data/corpus_BF.csv' # use your path
BF = pd.read_csv(filename, index_col=None, sep = ';', header=0,encoding='utf-8').reset_index()

# Lowercasing the text
BF['text_clean'] = BF['TXT'].apply(lambda x: x.lower())

BF_agg = BF.groupby(['ANNEE'])['index'].count().reset_index()
```

1. `pd.read_csv(filename, index_col=None, sep=';', header=0, encoding='utf-8').reset_index()` : Cette ligne lit les données du fichier CSV spécifié par `filename` dans un DataFrame Pandas (`BF`). Les paramètres utilisés

dans la fonction `pd.read_csv` sont les suivants:

- `index_col=None`: Spécifie qu'aucune colonne spécifique ne doit être utilisée comme index.
- `sep=';'`: Spécifie le séparateur utilisé dans le fichier CSV (dans ce cas, un point-virgule ';').
- `header=0`: Indique que la première ligne du fichier CSV contient les en-têtes des colonnes.
- `encoding='utf-8'`: Spécifie l'encodage des caractères du fichier en UTF-8.
- `.reset_index()`: Réinitialise l'index du DataFrame résultant.

2. `BF['text_clean'] = BF['TXT'].apply(lambda x: x.lower())`: Crée une nouvelle colonne 'text_clean' dans le DataFrame `BF` en appliquant une fonction lambda à la colonne 'TXT'. La fonction lambda convertit le texte dans 'TXT' en minuscules.
3. `BF_agg = BF.groupby(['ANNEE'])['index'].count().reset_index()`: Effectue une agrégation en regroupant le DataFrame `BF` en fonction de la colonne 'ANNEE' et en comptant le nombre d'occurrences ('index') dans chaque groupe. Le résultat est stocké dans le DataFrame 'BF_agg'.

L'exécution de ce code, renvoie le DataFrame traité ('BF') contenant les données du fichier CSV avec du texte en minuscules, et un DataFrame agrégé ('BF_agg') résumant les données par année ('ANNEE') et comptant le nombre d'occurrences.

- `index_col=None` : Indique que aucune colonne spécifique ne doit être utilisée comme index.
- `sep=';'` : Indique le séparateur utilisé dans le fichier CSV (dans ce cas, un point-virgule ';').
- `header=0` : Indique que la première ligne du fichier CSV contient les en-têtes de colonne.
- `encoding='utf-8'` : Spécifie l'encodage des caractères du fichier en UTF-8.
- `.reset_index()` : Réinitialise l'index du DataFrame résultant.

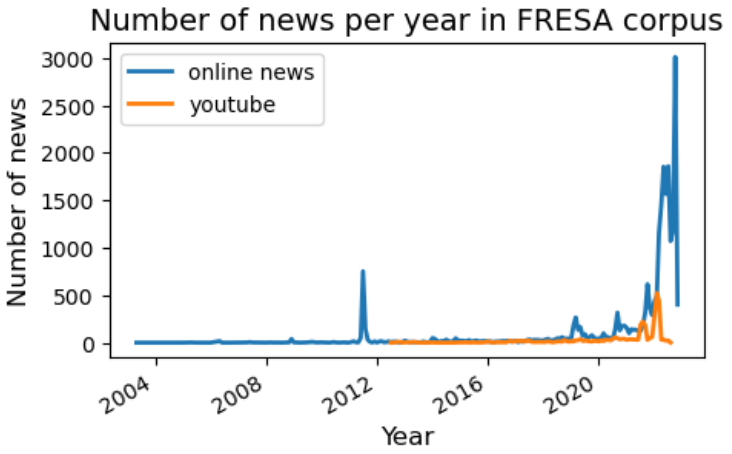
Timelines

```
label_size = 12
title_size = 14

fig, ax = plt.subplots(figsize=(5,3))
bp = FRESA_agg.groupby('source_norm').plot('year_month', 'index',ax=ax, marker='', linewidth=2,
                                           legend=False)
plt.title('Number of news per year in FRESA corpus', size = title_size)
plt.xlabel('Year',size=label_size)
plt.ylabel('Number of news', size = label_size)
plt.legend(labels = ['online news', 'youtube'], loc = 'upper left')
plt.show()
```

1. `label_size` et `title_size` : Ces variables stockent les tailles de police pour les étiquettes et les titres de votre graphique.
2. `fig, ax = plt.subplots(figsize=(5, 3))` : Cette ligne crée une figure Matplotlib (`fig`) et un axe (`ax`) avec une taille spécifiée (5 unités de largeur et 3 unités de hauteur).
3. `bp = FRESA_agg.groupby('source_norm').plot('year_month', 'index', ax=ax, marker='', linewidth=2, legend=False)` : Cette ligne regroupe le DataFrame `FRESA_agg` par la colonne 'source_norm' et crée un graphique en ligne pour chaque groupe (actualités en ligne et YouTube) en utilisant 'year_month' comme axe des x et 'index' comme axe des y. Le paramètre `ax=ax` spécifie l'axe à utiliser pour le graphique, et `marker=''` et `linewidth=2` contrôlent le style du marqueur et de la ligne pour le graphique. Le paramètre `legend=False` désactive la légende automatique.

4. `plt.title('Nombre de nouvelles par an dans le corpus FRESA', size=title_size)` : Définit le titre du graphique et la taille de la police.
5. `plt.xlabel('Année', size=label_size)` : Définit l'étiquette de l'axe des x et la taille de la police.
6. `plt.ylabel('Nombre de nouvelles', size=label_size)` : Définit l'étiquette de l'axe des y et la taille de la police.
7. `plt.legend(labels=['actualités en ligne', 'YouTube'], loc='upper left')` : Ajoute manuellement une légende au graphique avec des étiquettes spécifiées et une position (en haut à gauche).
8. `plt.show()` : Affiche le graphique.

Ce code crée un graphique en ligne montrant le nombre de nouvelles par an dans le corpus FRESA pour les sources d'actualités en ligne et YouTube, avec des axes étiquetés et une légende.

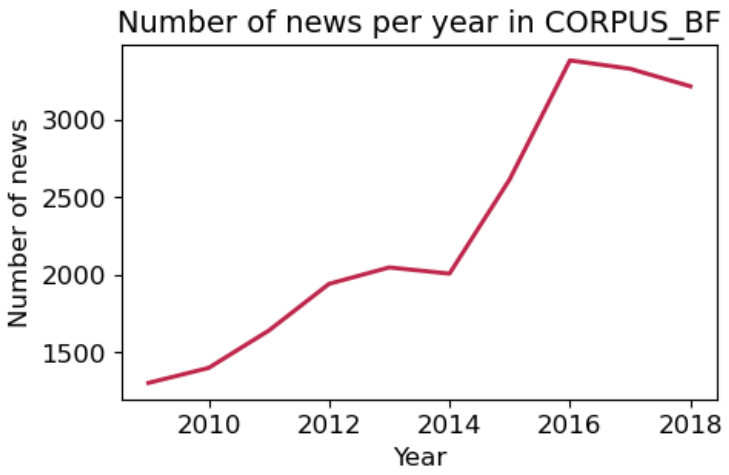



```
figsize(5,3)
plt.plot('ANNEE', 'index', data=BF_agg, marker='', color='#be254a', linewidth=2)
plt.xlabel('Year',size=label_size)
plt.ylabel('Number of news', size = label_size)
plt.title('Number of news per year in CORPUS_BF', size = title_size)
plt.xticks(size = label_size)
plt.yticks(size = label_size)
plt.show()
#plt.xticks(['12-31',  '01-05',  '01-09',  '01-13', '01-17', '01-21', '01-25','01-28'])
#plt.grid(True)
```

1. `figsize(5, 3)`: Cette ligne définit la taille de la figure à 5 unités de largeur et 3 unités de hauteur. Cependant, elle devrait être corrigée en `plt.figure(figsize=(5, 3))` pour définir correctement la taille de la figure.
2. `plt.plot('ANNEE', 'index', data=BF_agg, marker='', color='#be254a', linewidth=2)`: Cette ligne crée un graphique de ligne en utilisant la colonne 'ANNEE' comme axe des x et la colonne 'index' comme axe des y. Elle spécifie qu'aucun marqueur ne doit être utilisé, définit la couleur de la ligne à '#be254a' et ajuste la largeur de la ligne à 2.
3. `plt.xlabel('Année', size=label_size)`: Définit l'étiquette de l'axe des x et la taille de la police.
4. `plt.ylabel('Nombre de nouvelles', size=label_size)`: Définit l'étiquette de l'axe des y et la taille de la police.
5. `plt.title('Nombre de nouvelles par an dans CORPUS_BF', size=title_size)`: Définit le titre du graphique et la taille de la police.
6. `plt.xticks(size=label_size)`: Définit la taille de la police des étiquettes de l'axe des x.

7. `plt.yticks(size=label_size)` : Définit la taille de la police des étiquettes de l'axe des y.
8. `plt.show()` : Affiche le graphique.

Les lignes commentées avec `plt.xticks` et `plt.grid(True)` personnalisent l'apparence des graduations de l'axe des x et active une grille dans le graphique.



4. Coverage between lexicons and corpus

Corpus BF

Frequency of vocabulary with lexicon from Fraiberger et al.

```
txt = BF['text_clean'].tolist()
d = freq_lexicon(fraibeger_lexicon_fr, txt)
```

La fonction `freq_lexicon` analyse la fréquence des termes du lexique français (`fraibeger_lexicon_fr`) dans la colonne 'text_clean' du DataFrame 'BF' (`BF['text_clean']`). La fonction `freq_lexicon` renvoie un dictionnaire contenant les mots-clés correspondants et leur nombre d'occurrences.

1. `BF['text_clean']` : Accède à la colonne 'text_clean' du DataFrame 'BF', qui contient des données textuelles prétraitées en minuscules.
2. `.tolist()` : Convertit les valeurs de la colonne 'text_clean' en une liste Python nommée `txt`.
3. `freq_lexicon(fraibeger_lexicon_fr, txt)` : Appelle la fonction `freq_lexicon` avec la liste `fraibeger_lexicon_fr` des termes du lexique français et la liste `txt` des données textuelles prétraitées en tant qu'arguments. Cette fonction analyse la fréquence des termes du lexique dans le texte.

Le résultat (`d`) est un dictionnaire contenant les mots-clés correspondants et leurs comptages respectifs en fonction de leurs occurrences dans la colonne 'text_clean' du DataFrame 'BF'. On peut ensuite traiter davantage ce dictionnaire pour extraire des informations pertinentes sur les termes du lexique.

Most frequent keywords (term-frequency)

```
d['occurrences'][0:5]
```

[('corruption', 2973),
('conflit', 2056),
('siège', 1725),
('terroriste', 1279),
('réfugiés', 1128)]

Most frequent keywords (document-frequency)

```
d['doc'][0:5]
```

[('siège', 1184),
('conflit', 983),
('corruption', 787),
('terroriste', 687),
('inondations', 503)]

Most frequent categories (term-frequency)

```
f_occ = df_freq(d['occurrences']).merge(fraibeger_lexicon, on=["fr"])
f_occ.groupby(['category'])['freq'].agg("sum").sort_values(ascending=False)
```

1. `df_freq(d['occurrences'])`: On utilise la fonction `df_freq` pour créer un DataFrame avec les termes du lexique et leurs fréquences basées sur les occurrences de `d['occurrences']`.
2. `.merge(fraibeger_lexicon, on=["fr"])`: On fusionne les termes du lexique et leurs fréquences avec le DataFrame 'fraibeger_lexicon' basé sur la colonne 'fr' pour associer les termes du lexique à leurs catégories correspondantes.
3. `f_occ.groupby(['category'])['freq'].agg("sum").sort_values(ascending=False)`: On regroupe le DataFrame fusionné `f_occ` selon la colonne 'category' puis on calcule la somme des fréquences ('freq') dans chaque catégorie. Enfin, on trie les résultats dans l'ordre décroissant pour afficher les catégories avec les fréquences cumulées les plus élevées en haut.

Ce code fournit un résumé des fréquences des termes du lexique regroupées par catégorie, triées dans l'ordre décroissant de la fréquence totale dans chaque catégorie. Le résultat montre quelles catégories ont la fréquence totale la plus élevée de termes du lexique.

category	
conflicts and violence	40.031898
political instability	18.121322
forced displacement	11.257768
weather shocks	8.007480
other	5.477644
food crisis	4.603201
economic issues	4.399714
pests and diseases	3.349282
environmental issues	2.463840
humanitarian aid	1.688390
land-related issues	0.494968

agricultural production issues	0.104493

Most frequent categories (document-frequency)

```
f_doc = df_freq(d['doc']).merge(fraibeger_lexicon, on=["fr"])
f_doc.groupby(['category'])['freq'].agg("sum").sort_values(ascending=False)
```

On effectue une opération similaire à celle précédente, mais cette fois-ci on travaille avec la partie 'doc' du dictionnaire `d`, qui contient des informations sur les occurrences des termes lexicaux dans les documents.

1. `df_freq(d['doc'])` : On utilise la fonction `df_freq` pour créer un DataFrame avec les termes lexicaux et leurs fréquences en fonction des occurrences dans les documents de `d['doc']`.
2. `.merge(fraibeger_lexicon, on=["fr"])` : On fusionne les termes lexicaux et leurs fréquences avec le DataFrame 'fraibeger_lexicon' en fonction de la colonne 'fr' pour associer les termes lexicaux à leurs catégories correspondantes.
3. `f_doc.groupby(['category'])['freq'].agg("sum").sort_values(ascending=False)` : On regroupe le DataFrame fusionné `f_doc` par la colonne 'category' puis on calcule la somme des fréquences ('freq') dans chaque catégorie. Enfin, on trie les résultats par ordre décroissant pour afficher les catégories avec les fréquences cumulées les plus élevées en haut.

Ce code fournit un résumé des fréquences des termes lexicaux dans les documents, regroupées par catégorie et triées par ordre décroissant de fréquence totale dans chaque catégorie. Il permet d'identifier les catégories avec les fréquences totales les plus élevées de termes lexicaux dans les documents.

category	
conflicts and violence	45.410988
political instability	11.169080
weather shocks	8.283646
forced displacement	8.017462
food crisis	6.260647
economic issues	5.919932
other	5.249148
pests and diseases	3.204855
environmental issues	3.130324
humanitarian aid	2.448893
land-related issues	0.713373
agricultural production issues	0.191652

Frequency of vocabulary with lexicon from experts (Deleglise)

```
d2 = freq_lexicon(list(set(expert_C)), txt)
```

On utilise à nouveau la fonction `freq_lexicon`, cette fois avec un ensemble différent de termes lexicaux provenant de `expert_C`, et en analysant leur fréquence dans les données `txt`. Cela donne des informations sur la fréquence des termes du lexique `expert_C` dans nos données textuelles.

Le résultat, stocké dans `d2`, est un dictionnaire contenant les mots-clés correspondants et leurs comptes respectifs basés sur leurs occurrences dans les données `txt`.

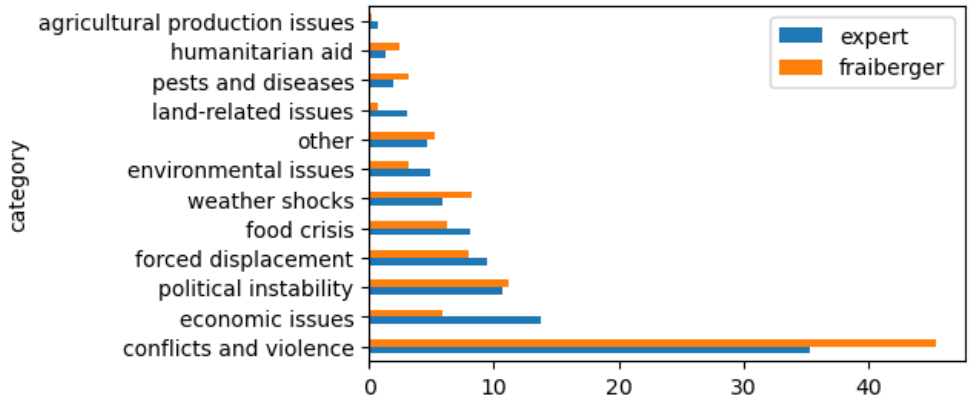
```
d2['occurrences'][0:5]
d2['doc'][0:5]

e_occ = df_freq(d2['occurrences']).merge(expert_lexicon_conv, on=["fr"])
e_occ.groupby(['category'])['freq'].agg("sum").sort_values(ascending=False)
```

Most frequent keywords (term-frequency)	Most frequent keywords (document-frequency)	Most frequent categories (term-frequency)
[('armée', 4122), ('conflit', 2056), ('attaque', 2018), ('pauvreté', 1568), ('insécurité', 1506)]	[('armée', 1310), ('attaque', 1086), ('pauvreté', 1013), ('conflit', 983), ('insécurité', 952)]	**conflicts and violence** 31.691488 **political instability** 17.152009 **economic issues** 9.946751 **food crisis** 8.989933 **forced displacement** 6.714369 **weather shocks** 5.898993 **environmental issues** 5.058657 **pests and diseases** 4.650969 **other** 4.526167 **land-related issues** 3.490307 **humanitarian aid** 1.368666 **agricultural production issues** 0.511690

```
e_doc = df_freq(d2['doc']).merge(expert_lexicon_conv, on=["fr"])
pd.DataFrame(e_doc.groupby(['category'])['freq'].agg("sum").sort_values(ascending=False))
```

conflicts and violence	35.275398
economic issues	13.741330
political instability	10.697674
forced displacement	9.506324
food crisis	8.135455
weather shocks	5.924113
environmental issues	4.887801
other	4.659323
land-related issues	3.059976
pests and diseases	1.982864
humanitarian aid	1.370869
agricultural production issues	0.758874

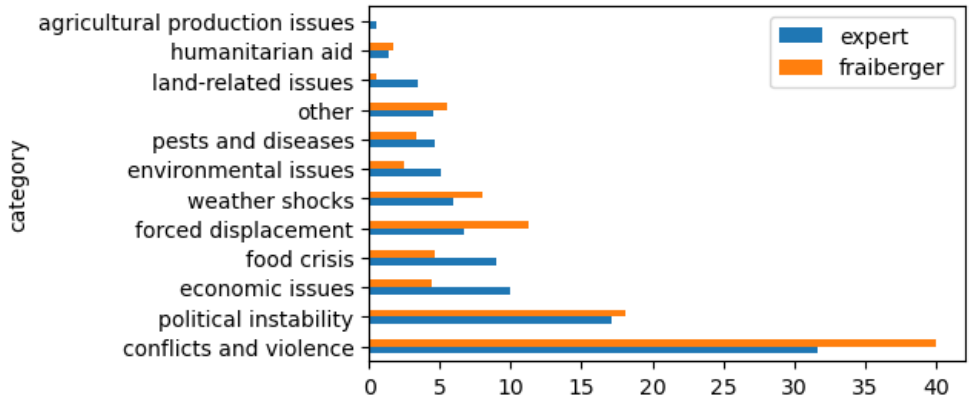
```
df1 = pd.DataFrame(e_doc.groupby(['category'])['freq'].agg("sum")).reset_index()
df1.columns = ['category', 'expert']
df2 = pd.DataFrame(f_doc.groupby(['category'])['freq'].agg("sum")).reset_index()
df2.columns = ['category', 'fraiberger']
dfall = df1.merge(df2, on = ['category'])
dfall = dfall.set_index('category')
dfall.sort_values(by = "expert", ascending=False).plot(kind='barh')
```



1. `pd.DataFrame(e_doc.groupby(['category'])['freq'].agg("sum")).reset_index()` : Cette ligne agrège la colonne 'freq' du DataFrame `e_doc` par 'category', calculant la somme des fréquences dans chaque catégorie, puis crée un nouveau DataFrame (`df1`) pour stocker les résultats. Le DataFrame résultant a deux colonnes : 'category' et 'expert'.
2. `pd.DataFrame(f_doc.groupby(['category'])['freq'].agg("sum")).reset_index()` : De manière similaire, cette ligne agrège la colonne 'freq' du DataFrame `f_doc` par 'category', calculant la somme des fréquences dans chaque catégorie, et crée un nouveau DataFrame (`df2`) pour stocker les résultats. Ce DataFrame a également deux colonnes : 'category' et 'fraiberger'.
3. `dfall = df1.merge(df2, on=['category'])` : On fusionne les deux DataFrames `df1` et `df2` sur la colonne 'category' pour combiner les informations de fréquence des deux lexiques.
4. `dfall = dfall.set_index('category')` : On définit la colonne 'category' comme index du DataFrame combiné `dfall` .
5. `dfall.sort_values(by="expert", ascending=False).plot(kind='barh')` : Enfin, on trie le DataFrame combiné `dfall` par la colonne 'expert' par ordre décroissant et on crée un graphique à barres horizontales (`barh`) pour visualiser les fréquences des catégories du lexique 'expert'. Les fréquences du lexique 'expert' sont utilisées comme référence pour le tri.

Ce code génère un graphique à barres horizontales qui compare les fréquences des catégories entre les lexiques 'expert' et 'fraiberger', triées par ordre décroissant selon les fréquences du lexique 'expert'. Chaque catégorie est représentée par une barre horizontale, et la longueur de la barre correspond à la fréquence de la catégorie dans le lexique 'expert'. Cette visualisation aide à comprendre l'importance relative des différentes catégories dans les deux lexiques, le lexique 'expert' servant de référence.

```
df1 = pd.DataFrame(e_occ.groupby(['category'])['freq'].agg("sum")).reset_index()
df1.columns = ['category', 'expert']
df2 = pd.DataFrame(f_occ.groupby(['category'])['freq'].agg("sum")).reset_index()
df2.columns = ['category', 'fraiberger']
dfall = df1.merge(df2, on = ['category'])
dfall = dfall.set_index('category')
dfall.sort_values(by = "expert", ascending=False).plot(kind='barh')
```



1. `pd.DataFrame(e_occ.groupby(['category'])['freq'].agg("sum")).reset_index()` : Agrège la colonne 'freq' du DataFrame `e_occ` par 'category', en calculant la somme des fréquences pour chaque catégorie, puis crée un nouveau DataFrame `df1` pour stocker les résultats. Le DataFrame résultant a deux colonnes : 'category' et 'expert'.
2. Renomme les colonnes dans `df1` en 'category' et 'expert' en utilisant `df1.columns`.
3. `pd.DataFrame(f_occ.groupby(['category'])['freq'].agg("sum")).reset_index()` : Similaire à l'étape précédente, agrège la colonne 'freq' du DataFrame `f_occ` par 'category', en calculant la somme des fréquences pour chaque catégorie, et crée un nouveau DataFrame `df2` pour stocker les résultats. Ce DataFrame a également deux colonnes : 'category' et 'fraiberger'.
4. Renomme les colonnes dans `df2` en 'category' et 'fraiberger' en utilisant `df2.columns`.
5. `df1.merge(df2, on=['category'])` : Fusionne les deux DataFrames `df1` et `df2` sur la colonne 'category' pour combiner les informations de fréquence des deux lexiques.
6. `dfall = dfall.set_index('category')` : Définit la colonne 'category' comme index du DataFrame combiné `dfall`.
7. `dfall.sort_values(by="expert", ascending=False).plot(kind='barh')` : Trie le DataFrame combiné `dfall` selon la colonne 'expert' par ordre décroissant et crée un graphique à barres horizontales (`barh`) pour visualiser les fréquences des catégories. Les fréquences du lexique 'expert' sont utilisées comme référence pour le tri.

Ce code génère un graphique à barres horizontales comparant les fréquences des catégories entre les lexiques 'expert' et 'fraiberger', avec les fréquences du lexique 'expert' comme référence. Chaque catégorie est représentée par une barre horizontale, et la longueur de la barre correspond à la fréquence de la catégorie dans le lexique 'expert'. Cette visualisation permet de comprendre l'importance relative des différentes catégories dans les deux lexiques.

💡 La principale différence réside dans les DataFrames sources utilisés pour l'agrégation :

- Dans le **code précédent (Code 1)**, on agrège des données à partir de `e_doc` et `f_doc`, qui contiennent des fréquences de termes lexicaux basées sur leur occurrence dans les documents.
- Dans le **code actuel (Code 2)**, on agrège des données à partir de `e_occ` et `f_occ`, qui contiennent probablement des fréquences de termes lexicaux basées sur leur occurrence mais ne sont pas précisées comme étant liées aux documents.

Mis à part cela, les structures de code sont presque identiques, et les deux codes créent des diagrammes à barres horizontaux comparant les fréquences des catégories entre les lexiques 'expert' et 'fraiberger', les fréquences du lexique 'expert' étant utilisées comme référence pour le tri.

Corpus FRESA

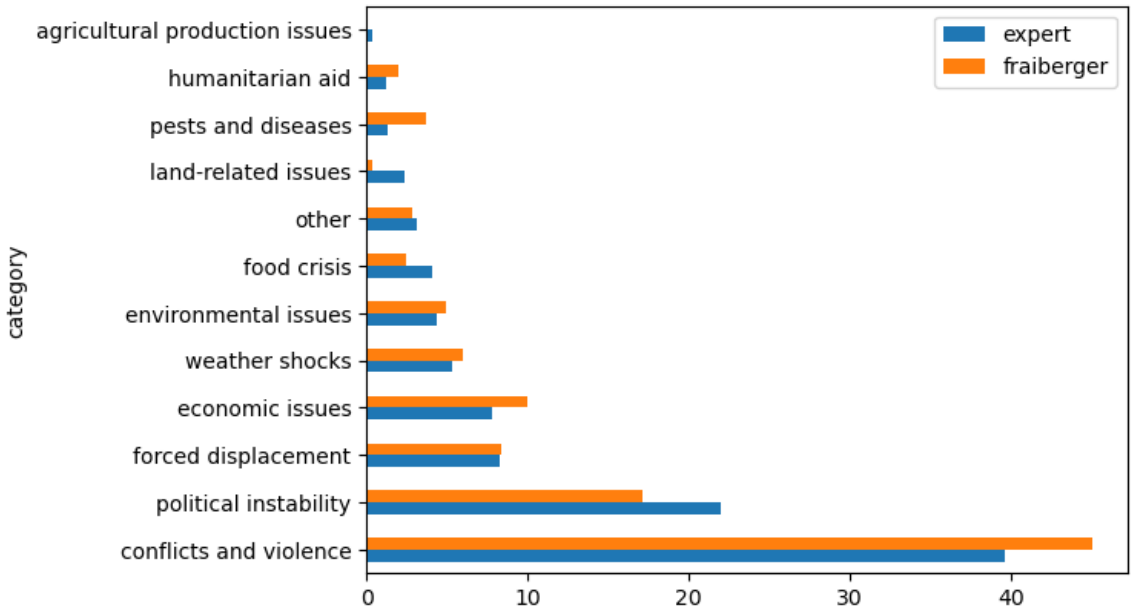
```
txt = FRESA['text_clean'].tolist()
df = freq_lexicon(fraibeger_lexicon_fr, txt)
de = freq_lexicon(list(set(expert_C)), txt)

e_occ = df_freq(de['occurrences']).merge(expert_lexicon_conv, on=["fr"])
e_doc = df_freq(de['doc']).merge(expert_lexicon_conv, on=["fr"])
f_occ = df_freq(df['occurrences']).merge(fraibeger_lexicon, on=["fr"])
f_doc = df_freq(df['doc']).merge(fraibeger_lexicon, on=["fr"])

f_occ = df_freq(df['occurrences']).merge(fraibeger_lexicon, on=["fr"])
f_doc = df_freq(df['doc']).merge(fraibeger_lexicon, on=["fr"])

df1 = pd.DataFrame(e_occ.groupby(['category'])['freq'].agg("sum")).reset_index()
df1.columns = ['category', 'expert']
df2 = pd.DataFrame(f_occ.groupby(['category'])['freq'].agg("sum")).reset_index()
df2.columns = ['category', 'fraiberger']
dfall = df1.merge(df2, on = ['category'])
dfall = dfall.set_index('category')
dfall.sort_values(by = "expert", ascending=False).plot(kind='barh')
```

On effectue une analyse lexicale et une visualisation basées sur les occurrences lexicales dans les données textuelles du DataFrame 'FRESA' et de deux lexiques différents ('fraibeger_lexicon_fr' et 'expert_C').

1. On extraie la colonne 'text_clean' du DataFrame 'FRESA' et la convertie en une liste Python nommée `txt`, qui contient des données textuelles prétraitées.
2. En utilisant la fonction `freq_lexicon`, on analyse la fréquence des termes lexicaux français ('fraibeger_lexicon_fr') dans les données `txt`, et le résultat est stocké dans le dictionnaire `df`.
3. On utilise de manière similaire la fonction `freq_lexicon` pour analyser la fréquence des termes lexicaux de 'expert_C' dans les données `txt`, et le résultat est stocké dans le dictionnaire `de`.
4. On calcule séparément la fréquence des occurrences lexicales (`e_occ` et `f_occ`) et des documents (`e_doc` et `f_doc`) pour les lexiques 'expert' et 'fraiberger', puis on fusionne ces informations avec leurs mappings de catégories lexicales respectives.
5. On crée deux DataFrames (`df1` et `df2`) en regroupant les informations de fréquence par catégorie et en additionnant les fréquences au sein de chaque catégorie. Ces DataFrames ont les colonnes 'category', 'expert' et 'fraiberger'.
6. On fusionne `df1` et `df2` sur la colonne 'category' pour créer `dfall`, qui contient les informations de fréquence agrégées pour les deux lexiques, indexées par catégorie.

7. On définit la colonne 'category' comme index du DataFrame `dfall`.
8. Enfin, on crée un graphique à barres horizontales (`barh`) pour visualiser les fréquences agrégées des catégories des deux lexiques, triées par ordre décroissant des fréquences du lexique 'expert'.

Cette visualisation vous permet de comparer et de comprendre l'importance relative des différentes catégories dans les lexiques 'expert' et 'fraiberger'.




```
df1 = pd.DataFrame(e_doc.groupby(['category'])['freq'].agg("sum")).reset_index()
df1.columns = ['category', 'expert']
df2 = pd.DataFrame(f_doc.groupby(['category'])['freq'].agg("sum")).reset_index()
df2.columns = ['category', 'fraiberger']
dfall = df1.merge(df2, on = ['category'])
dfall = dfall.set_index('category')
dfall.sort_values(by = "expert", ascending=False).plot(kind='barh')
```

1. `pd.DataFrame(e_doc.groupby(['category'])['freq'].agg("sum")).reset_index()` : Cette ligne regroupe la colonne 'freq' du DataFrame `e_doc` par 'category', calcule la somme des fréquences pour chaque catégorie, et crée un nouveau DataFrame `df1` pour stocker les résultats. Le DataFrame résultant a deux colonnes : 'category' et 'expert'.
2. Renomme les colonnes dans `df1` en 'category' et 'expert' en utilisant `df1.columns`.
3. `pd.DataFrame(f_doc.groupby(['category'])['freq'].agg("sum")).reset_index()` : De manière similaire, cette ligne regroupe la colonne 'freq' du DataFrame `f_doc` par 'category', calcule la somme des fréquences pour chaque catégorie, et crée un nouveau DataFrame `df2` pour stocker les résultats. Ce DataFrame a également deux colonnes : 'category' et 'fraiberger'.
4. Renomme les colonnes dans `df2` en 'category' et 'fraiberger' en utilisant `df2.columns`.
5. `df1.merge(df2, on=['category'])` : Fusionne les deux DataFrames `df1` et `df2` sur la colonne 'category' pour combiner les informations de fréquence des deux lexiques.
6. `dfall = dfall.set_index('category')` : Définit la colonne 'category' comme l'index du DataFrame combiné `dfall`.

7. `dfall.sort_values(by="expert", ascending=False).plot(kind='barh')` : Trie le DataFrame combiné `dfall` par la colonne 'expert' par ordre décroissant et crée un diagramme à barres horizontales (`barh`) pour visualiser les fréquences des catégories. Les fréquences du lexique 'expert' sont utilisées comme référence pour le tri.

Ce code génère un diagramme à barres horizontales qui compare les fréquences des catégories entre le lexique 'expert' et le lexique 'fraiberger', avec les fréquences du lexique 'expert' utilisées comme référence pour le tri. Chaque catégorie est représentée par une barre horizontale, et la longueur de la barre correspond à la fréquence de la catégorie dans le lexique 'expert'. Cette visualisation permet de comprendre l'importance relative des différentes catégories dans les deux lexiques.