Tutorial 2: Wordnet

(Syn-)Semantische Netze wie WordNet stellen wichtige Ressourcen für NLP dar. In diesem Tutorial werden sie grundlegende Funktionalitäten von WordNet und der deutschen Variante GermaNet anwenden. Nutzen Sie die Dokumentationen zu WordNet (https://www.nltk.org/howto/wordnet.html (https://www.nltk.org/howto/wordnet.html)) und GermaNet (https://germanetpy.readthedocs.io/en/latest/ (https://germanetpy.readthedocs.io/en/latest/)). Sollten Sie mit der API Dokumentation nicht weiterkommen, nutzen Sie weitere Dokumentationen und Hilfestellungen wie stackoverflow.com oder Tutorials.

Aufgabe 1: Importieren der Module und Daten

a: Import von NLP Modulen

Importieren Sie wie im ersten Tutorial Pandas, Numpy, NLTK und RE und importieren sie WordNet (als wn) von nltk.corpus.

b: Import von "Quality-of-Life Modulen"

Oft sind Module nicht notwendig, erleichtern aber die Arbeit mit größeren Korpora. Installieren sie pandarallel. Importieren Sie wie im ersten Tutorial die Methode pandarallel (für Parallelization in Pandas) und initialisieren Sie diese mit pandarallel.initialize(). Sie können nun bei parallelisierbaren Aufgaben parallel_apply() anstelle der Pandas Methode apply() verwenden.

In [1]:

```
import pandas as pd
from nltk.corpus import wordnet as wn

import pandas as pd
print ("pandas", pd.__version__)

import numpy as np
print ("numpy", np.__version__)

import nltk
print ("nltk", nltk.__version__)

import re
print ("re", re.__version__)

from pandarallel import pandarallel  # parallelization
pandarallel.initialize()
```

```
pandas 1.0.5
numpy 1.18.5
nltk 3.5
re 2.2.1
INFO: Pandarallel will run on 8 workers.
INFO: Pandarallel will use Memory file system to transfer data betwe
en the main process and workers.
```

1.2: Importieren der Daten

Für das WordNet Tutorial ist ein Datensatz zu Biased Words (Wörtern die Voreingenommenheit oder unsachliche Wertung tragen) zur Verfügung gestellt. Importieren Sie das als pickle gespeicherte Dataframe "data.pkl".

In [2]:

```
data = pd.read_pickle("data.pkl")
```

In [3]:

```
data
```

Out[3]:

	sentence	topic	Label_bias	biased_words
0	YouTube is making clear there will be no “birt...	elections-2020	Biased	[belated, birtherism]
1	The increasingly bitter dispute between Americ...	sport	Non-biased	[bitter]
2	So while there may be a humanitarian crisis dr...	immigration	Biased	[crisis]
3	A professor who teaches climate change classes...	environment	Non-biased	[legitimate]
4	Looking around the United States, there is nev...	abortion	Biased	[killing, never, developing, humans, enough]
...	...	...	...	...
1695	In every case legislators are being swarmed by...	gender	Biased	[deceit, hysteria, swarmed, right-wing]
1696	Polls show the transgender ideology is deeply ...	gender	Biased	[ideology, unpopular]
1697	Democrats and Republicans stood and applauded ...	gender	Non-biased	[saluted]
1698	As a self-described Democratic socialist, Sen....	middle-class	Non-biased	[outspoken]
1699	During the segment, Colbert also bemoaned the ...	white-nationalism	Non-biased	[bemoaned]

1700 rows × 4 columns

Aufgabe 2: Synsets

Synsets sind die bedeutungsunterscheidbaren Definitionen von Wörtern bzw. Tokens. Ein Anwendungszweck ist es, den Kontext zu Wörtern um bedeutungsgleiche Wörter zu erweitern. Für den gegebenen Datensatz möchten wir die Synonyme zu den Biased Words in Form der \textit{Lemmas} zu den zugehörigen Synsets erfassen.

a: Synsets finden

Extrahieren Sie eine Liste der paarweise verschiedenen biased words (Spalte "Label_bias") aus dem Datensatz und speichern Sie diese in einer separaten Liste "b_words" (nicht im Dataframe!).

In [4]:

```
b_words = list(set([a for b in data.biased_words.tolist() for a in b]))
```

b: Synonyme finden

Für jedes der Wörter, bestimmen sie alle Synsets und alle zu den Synsets gehörigen Lemmas. Speichern Sie alle paarweise verschiedenen Lemmas zu allen biased Words in einer Liste "b_words_synonyms".

In [5]:

```
synonyms = []
for word in b_words:
    for syn in wn.synsets(word):
         for lemma in syn.lemmas():
                synonyms.extend([lemma.name()])

b_words_synonyms = list(set(synonyms))
```

2.3 Kandidaten für weitere biased Words

Die soeben bestimmten Synonyme zu den Biased Words stellen einen guten Startpunkt für die manuelle Bestimmung von weiteren Biased Words dar. Erstellen Sie abschließend eine Liste "new_b_words", in der Sie alle Wörter der Liste "b_words_synonyms" speichern, die nicht in der ursprünglichen Liste von "b_words" enthalten sind. Lassen Sie sich für alle 3 erzeugten Listen die Anzahl der enthaltenen Wörter ausgeben.

In [6]:

```
new_b_words = [a for a in b_words_synonyms if a not in b_words]
```

In [7]:

```
print(len(b_words))
print(len(b_words_synonyms))
print(len(new_b_words))
```

```
2258
10067
8512
```

Teil 2 GermaNet

GermaNet ist die an der Uni Tübingen entwickelte Version von Germanet für ralationale synsemantische Wortnetze in deutscher Sprache. Obwohl die grundlegenden Funktionalitäten weitestgehend identisch zu Wordnet sind, werden sie anders aufgerufen. Da Sie sich im weiteren Verlauf dieser Veranstaltung vermehrt mit den Datensätzen aus dem ESUPOL Projekt beschäftigen werden, könnte GermaNet ein sinnvolles Tool für semantische Analysen darstellen.

a: Germanet installieren und importieren

GermaNet ist nicht öffentlich zugänglich, die TH Köln hat eine Lizenz für die Verwendung im Rahmen von Lehre und Forschung zur Verfügung gestellt bekommen. Nutzen Sie GermaNet daher nur für Aufgaben und Projekte im Rahmen dieser Lehrveranstaltung. Kopieren Sie den bereitgestellen Ordner "germanetpy" in ihren site-packages Ordner. Vergewissern Sie sich, dass Germanet ordnungsgemäß gefunden wird. Sie können sicher gehen und das Modul noch einmal über pip installieren:

```
import sys
!{sys.executable} -m pip install -U germanetpy
```

WordNet nutzt XML für die Relationen und Textdateien für Frequencies, die an einem bestimmten Ort abgelegt sein müssen. Folgen sie den Vorgaben der offiziellen API um Germanet richtig einzurichten:

```
from pathlib import Path
from germanetpy.germanet import Germanet

data_path = str(Path.home()) + "/germanet/GN_V150/GN_V150_XML"
frequencylist_nouns = str(Path.home()) + "/germanet/GN_V150/FreqLists/noun_freqs_decow14_16.txt"
germanet = Germanet(data_path)
```

In [8]:

```
from pathlib import Path
from germanetpy.germanet import Germanet

data_path = str(Path.home()) + "/germanet/GN_V150/GN_V150_XML"
frequencylist_nouns = str(Path.home()) + "/germanet/GN_V150/FreqLists/noun_freqs_decow14_16.txt"
germanet = Germanet(data_path)
```

```
Load GermaNet data...: 100%|██████████| 99.99999999999996/100 [00:10
<00:00,  9.73it/s]
Load Wictionary data...: 100%|██████████| 100.0/100 [00:00<00:00, 57
3.59it/s]
Load Ili records...: 100%|██████████| 100.0/100 [00:00<00:00, 19897
0.78it/s]
```

b: Datensatz importieren

Importieren sie die Datei "single_term_suggestions.txt" als Dataframe. Die Datei enthält eine Liste von single-Word Query Suggestions aus dem in der Vorlesung vorgestellten Datensatz zur Bundestagswahl 2017. Sie können die Pandas-Methode "read_csv" nutzen.

In [9]:

```
data_ger = pd.read_csv("single_term_suggestions.txt")
```

Aufgabe 4: Germanet nutzen

a: Synsets

Bestimmen Sie jeweils für alle Suggestions (also jede Zeile der Daten) alle Synsets und speichern Sie die Liste in einer separaten Spalte.

In [10]:

```
data_ger["synsets_ger"] = data_ger.apply(lambda row: germanet.get_synsets_by_o
rthform(row["suggestion_ger"], ignorecase = True), axis=1)
```

b: Lexikalische Einheiten

Bestimmen sie für jede Zeile für alle Synsets jeweils alle Lemmas (lexikalischen Einheiten, also lexunits) und tragen sie diese als eine Liste in eine neue Spalte ein.

In [11]:

```
def get_names_synsets(syns):
    ret = []
    try:
        for syn in syns:
            for lemma in syn.lexunits:
                ret.extend(lemma.get_all_orthforms())
    #print(ret)
        return ret
    except:
        return ret
```

In [12]:

```
data_ger["lexunits"] = data_ger.apply(lambda row: get_names_synsets(row["synse
ts_ger"]), axis=1)
```

c: Hypernyms

Semantische Netze wie Germanet beschreiben Ist-Beziehungen zwischen Synsets. Hypernyme (Übertypen) und Hyponyme (Untertypen) können hilfreich für die Klassifizierung von Begriffen sein. Bestimmen Sie für alle Synsets jeder Suggestion jeweils alle Hypernyms und speichern sie deren Synsets in einer Spalte "hypernyms". Bestimmen sie anschließend die Lemmas dieser Hypernyme und speichern Sie diese in einer separaten Spalte.

In [13]:

```
def get_hypernyms_from_list_of_synsets(list_syns):
    for syn in list_syns:
        return syn.direct_hypernyms
```

In [14]:

```
data_ger["hypernyms"] = data_ger.apply(lambda row: get_hypernyms_from_list_of_
synsets(row["synsets_ger"]), axis=1)
```

In [15]:

```
data_ger["lexunits_hypernyms"] = data_ger.apply(lambda row: get_names_synsets(
row["hypernyms"]), axis=1)
```

d: Hyponyms

Gehen sie wie in c vor, nur bestimmen Sie dieses Mal die Hyponyme der Suggestions und deren Lemmas.

In [16]:

```
def get_hyponyms_from_list_of_synsets(list_syns):
    for syn in list_syns:
        return syn.direct_hyponyms
```

In [17]:

```
data_ger["hyponyms"] = data_ger.apply(lambda row: get_hyponyms_from_list_of_sy
nsets(row["synsets_ger"]), axis=1)
```

In [18]:

```
data_ger["lexunits_hyponyms"] = data_ger.apply(lambda row: get_names_synsets(r
ow["hyponyms"]), axis=1)
```

In [19]:

```
data_ger[["suggestion_ger","lexunits_hypernyms", "lexunits_hyponyms", "lexunits"]]
```

Out[19]:

	suggestion_ger	lexunits_hypernyms	lexunits_hyponyms	lexunits
0	aa	[Fäkalien]	[Mist, Dung, Klabusterbeere, Kinderkacke, Losu...	[Kot, Scheiße, Exkrement, Stuhl, Kacke, Kaka, ...
1	aach	[]	[]	[]
2	aalten	[]	[]	[]
3	aarburg	[]	[]	[]
4	aaronn	[]	[]	[]
...	...	...	...	...
3757	zwangsdienst	[Dienst]	[]	[Zwangsdienst]
3758	zwangshypothek	[]	[]	[]
3759	zweibruecken	[]	[]	[]
3760	zwickau	[Stadt]	[]	[Zwickau]
3761	zwillinge	[Sternzeichen, Sternbild, Tierkreiszeichen, Haus]	[]	[Zwillinge]

3762 rows × 4 columns

Lassen Sie sich die Anzahl aller paarweise verschiedenen Hypernyme, Hyponyme und Lemmas ausgeben.

In [20]:

```
lexunits_hypernyms = list(set([a for b in data_ger.lexunits_hypernyms for a in b]))
lexunits_hyponyms = list(set([a for b in data_ger.lexunits_hyponyms for a in b]))
lexunits = list(set([a for b in data_ger.lexunits for a in b]))
```

In [21]:

```
print(len(lexunits_hypernyms))
print(len(lexunits_hyponyms))
```

```
2020
12417
```

e: Klassifizierungs-Tags mit Synsets

Synsemantische Netze können beispielsweise genutzt werden, um Begriffe zu klassifizieren. Nutzen Sie die identifizierten Hypernyme, um alle Städte in dem Dataset zu finden. Klassifizieren Sie in einer Spalte "location" alle Städte, Länder und Orte als "True" und alle anderen Suggestions als "False". Lassen sie sich ein Sub-Dataframe aller Locations im Datensatz ausgeben.

In [22]:

```
def is_word_in_list(liste, words):
    for word in words:
        if word in liste:
            return True
    return False
```

In [23]:

```
data_ger["city"] = data_ger.apply(lambda row: is_word_in_list(row["lexunits_hypernyms"], ["Land", "Dorf", "Stadt", "Ort", "Platz", "Staat", "Bundesland" ]), axis=1)
```

In [24]:

```
data_ger.loc[(data_ger["city"]==True)]
```

Out[24]:

	suggestion_ger	synsets_ger	lexunits	hypernyms	lexunits_hypernyr
43	afghanistan	[Synset(id=s44583, lexunits=Afghanistan)]	[Afghanistan]	{Synset(id=s44177, lexunits=Land, Staat)}	[Land, Sta
64	albanien	[Synset(id=s44497, lexunits=Albanien)]	[Albanien]	{Synset(id=s44177, lexunits=Land, Staat)}	[Land, Sta
69	aleppo	[Synset(id=s73659, lexunits=Aleppo)]	[Aleppo]	{Synset(id=s43645, lexunits=Stadt)}	[Sta
99	altenburg	[Synset(id=s73693, lexunits=Altenburg)]	[Altenburg]	{Synset(id=s43645, lexunits=Stadt)}	[Sta
121	amorbach	[Synset(id=s44036, lexunits=Amorbach)]	[Amorbach]	{Synset(id=s43645, lexunits=Stadt)}	[Sta
...	...	...	...	...	
3666	wittenberg	[Synset(id=s44111, lexunits=Wittenberg)]	[Wittenberg]	{Synset(id=s43645, lexunits=Stadt)}	[Sta
3691	worms	[Synset(id=s44019, lexunits=Worms)]	[Worms]	{Synset(id=s43645, lexunits=Stadt)}	[Sta
3702	wuppertal	[Synset(id=s44061, lexunits=Wuppertal)]	[Wuppertal]	{Synset(id=s43645, lexunits=Stadt)}	[Sta
3707	xanten	[Synset(id=s44082, lexunits=Xanten)]	[Xanten]	{Synset(id=s43645, lexunits=Stadt)}	[Sta
3760	zwickau	[Synset(id=s44112, lexunits=Zwickau)]	[Zwickau]	{Synset(id=s43645, lexunits=Stadt)}	[Sta

194 rows × 8 columns

BONUSAUFGABE: Semantische Ähnlichkeit und Verwandschaft

Über die relationale Struktur von Synsets lässt sich die Ähnlichkeit bzw. Verwandtschaft zweier Begriffe gleicher Wortart ableiten. Die Ähnlichkeit kann wiederum zum Beispiel zur Beseitigung von Ambiguität verwendet werden. Im Fall des Datensatzes bilden die Terme Suchvorschläge zu personenbezogenen Suchen in Suchmaschienen, denen die Namen von Politikern als Suchterm zugrunde liegen. Der Term "Abbruch" wurde also als Suchvorschlag für mindestens einen Namen eines Politikers vorgeschlagen. Um von den Suggestions nun jeweils das relevante Synset zu identifizieren, kann die Ähnlichkeit zum Synset "Politiker" (Synset(id=s34818, lexunits=Politiker, Politikerin)) bestimmt werden.

Gehen Sie wie im offiziellen Tutorial zu GermaNet (https://github.com/Germanet-sfs/germanetTutorials/tree/master/pythonAPI} (https://github.com/Germanet-sfs/germanetTutorials/tree/master/pythonAPI})) erläutert vor, um für alle Synsets aller Suggestions jeweils die Similarity zum Politiker Synset zu berechnen und speichern Sie das Synsets mit der höchsten Ähnlichkeit in einer Spalte "best_syn". Verwenden Sie entweder Path- oder IC- basierte Ähnlichkeit oder führen Sie beides separat durch. Exportieren Sie abschließend ein Sub-Dataframe, in dem nur solche Zeilen enthalten sind, deren Suggestion über mindestens 2 Synsets verfügt.

In [25]:

```
data_ger.at[1, "synsets_ger"]==True
```

Out[25]:

```
False
```

In [26]:

```
data_ger["synsets_check"] = data_ger.apply(lambda row: len(row["synsets_ger"
]), axis=1)
```

In [27]:

```
data_ger
```

Out[27]:

	suggestion_ger	synsets_ger	lexunits	hypernyms	lexunits_hy
0	aa	[Synset(id=s26358, lexunits=Kot, Scheiße, Exkr...	[Kot, Scheiße, Exkrement, Stuhl, Kacke, Kaka, ...	{Synset(id=s26356, lexunits=Fäkalien)}	
1	aach	[]	[]	None	
2	aalten	[]	[]	None	
3	aarburg	[]	[]	None	
4	aaronn	[]	[]	None	
...	...	...	...	...	
3757	zwangsdienst	[Synset(id=s131522, lexunits=Zwangsdienst)]	[Zwangsdienst]	{Synset(id=s19797, lexunits=Dienst)}	
3758	zwangshypothek	[]	[]	None	
3759	zweibruecken	[]	[]	None	
3760	zwickau	[Synset(id=s44112, lexunits=Zwickau)]	[Zwickau]	{Synset(id=s43645, lexunits=Stadt)}	
3761	zwillinge	[Synset(id=s29525, lexunits=Zwillinge)]	[Zwillinge]	{Synset(id=s29522, lexunits=Sternzeichen, Ster...	[Ster Tierkre

3762 rows × 9 columns

In [28]:

```
data_bonus = data_ger.loc[(data_ger["synsets_check"]>1)]
```

In [29]:

```
data_bonus
```

Out[29]:

	suggestion_ger	synsets_ger	lexunits	hypernyms	lexunits_
0	aa	[Synset(id=s26358, lexunits=Kot, Scheiße, Exkr...	[Kot, Scheiße, Exkrement, Stuhl, Kacke, Kaka, ...	{Synset(id=s26356, lexunits=Fäkalien)}	
6	abbruch	[Synset(id=s106285, lexunits=Abbruch), Synset(...	[Abbruch, Abbruch, Beendigung, Beenden, Aufhör...	{Synset(id=s17129, lexunits=Schaden, Schädigung)}	
15	abnehmen	[Synset(id=s52508, lexunits=wegnehmen, abnehme...	[wegnehmen, abnehmen, fortnehmen, abchecken, a...	{Synset(id=s52497, lexunits=nehmen)}	
18	abschied	[Synset(id=s17481, lexunits=Abschied, Lebewohl...	[Abschied, Lebewohl, Verabschiedung, Abschied]	{Synset(id=s17478, lexunits=Trennung)}	
26	adel	[Synset(id=s32247, lexunits=Adelstitel, Adel, ...	[Adelstitel, Adel, Adelsbezeichnung, Adelsgesc...	{Synset(id=s32215, lexunits=Titel)}	
...	...	...	...	...	
3741	zirkus	[Synset(id=s108415, lexunits=Zirkus), Synset(i...	[Circus, Zirkus, Circus, Zirkus, Zirkusunterne...	{Synset(id=s42745, lexunits=Arena, Stadion)}	[Are
3742	zitat	[Synset(id=s32274, lexunits=Zitat), Synset(id=...	[Zitat, Zitat]	{Synset(id=s32267, lexunits=Redewendung, Idiom...	[Re Idiom
3752	zug	[Synset(id=s76189, lexunits=Zug), Synset(id=s4...	[Zug, Luftzug, Zug, Luft, Gesichtszug, Zug, Zu...	{Synset(id=s24150, lexunits=Formation, Gruppie...	
3753	zukunft	[Synset(id=s51054, lexunits=Zukunft, Vorzeitig...	[Zukunft, Vorzeitigkeit, Hinkunft, Futur, Zuku...	{Synset(id=s51051, lexunits=Zeitstufe)}	
3755	zusammenbruch	[Synset(id=s17538, lexunits=Kollaps, Zusammenb...	[Kollaps, Zusammenbruch, Zusammenbruch, Crash]	{Synset(id=s17096, lexunits=Vorfall, Begebenhe...	[Vorfall, E

451 rows × 9 columns

In [30]:

```
from germanetpy.path_based_relatedness_measures import PathBasedRelatedness
from germanetpy.synset import WordCategory
from germanetpy.icbased_similarity import ICBasedSimilarity

relatedness_nouns = ICBasedSimilarity(germanet=germanet,
                                      wordcategory=WordCategory.nomen,
                                      path=frequencylist_nouns)
```

```
file /home/fabian/germanet/GN_V150/FreqLists/noun_freqs_decow14_16.t
xt does not exist
```

In [31]:

```
syn_politikerIn = germanet.get_synsets_by_orthform("Politiker")
syn_politikerIn
```

Out[31]:

```
[Synset(id=s34818, lexunits=Politiker, Politikerin)]
```

In [32]:

```
def find_distance_to_politician(synset):
    syn_p = germanet.get_synsets_by_orthform("Politiker").pop()
    path_distance = synset.shortest_path_distance(syn_p)
```

In [33]:

```
def find_most_related_to_politician_path(list_syns):
    # First, construct a path-based similarity object.
    # The johannis_wurm and leber_trans synsets are maximally far apart among
 nouns:
    johannis_wurm = germanet.get_synset_by_id("s49774")
    leber_trans = germanet.get_synset_by_id("s83979")
    relatedness_calculator = PathBasedRelatedness(germanet=germanet, category=
WordCategory.nomen, max_len=35, max_depth=20, synset_pair=(johannis_wurm, lebe
r_trans))

    syn_p = germanet.get_synsets_by_orthform("Politiker").pop()

    res = {}
    for syn in list_syns:
        if syn.word_category == WordCategory.nomen:
            res[syn] = relatedness_calculator.simple_path(syn, syn_p)
    if(len(res)>0):
        return max(res, key=res.get)
    else:
        return "no nouns"
```

In [39]:

```
def find_most_related_to_politician_ic(list_syns):
    syn_p = germanet.get_synsets_by_orthform("Politiker").pop()
    res = {}
    for syn in list_syns:
        if syn.word_category == WordCategory.nomen:
            res[syn] = relatedness_nouns.resnik(syn, syn_p, normalize=True)
    if(len(res)>0):
        return max(res, key=res.get)
    else:
        return "no nouns"
```

In [45]:

```
data_bonus["best_syn"] = data_bonus.apply(lambda row: find_most_related_to_pol
itician_path(row["synsets_ger"]) ,axis=1)
data_bonus["best_syn_ic"] = data_bonus.apply(lambda row: find_most_related_to_
politician_ic(row["synsets_ger"]) ,axis=1)
```

```
---------------------------------------------------------------------
-------
KeyError                                  Traceback (most recent cal
l last)
<ipython-input-45-10ed1a99a639> in <module>
----> 1 data_bonus.loc[["best_syn"]] = data_bonus.apply(lambda row:
 find_most_related_to_politician_path(row["synsets_ger"]) ,axis=1)
      2 data_bonus["best_syn_ic"] = data_bonus.apply(lambda row: fin
d_most_related_to_politician_ic(row["synsets_ger"]) ,axis=1)

~/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py in _
_setitem__(self, key, value)
    668         else:
    669             key = com.apply_if_callable(key, self.obj)
--> 670         indexer = self._get_setitem_indexer(key)
    671         self._setitem_with_indexer(indexer, value)
    672

~/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py in _
get_setitem_indexer(self, key)
    655         axis = self.axis or 0
    656         try:
--> 657             return self._convert_to_indexer(key, axis=axis)
    658         except TypeError as e:
    659

~/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py in _
convert_to_indexer(self, obj, axis, raise_missing)
   1730             else:
   1731                 # When setting, missing keys are not allowe
d, even with .loc:
-> 1732                 return self._get_listlike_indexer(obj, axis,
raise_missing=True)[1]
   1733         else:
   1734             try:

~/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py in _
get_listlike_indexer(self, key, axis, raise_missing)
   1550             keyarr, indexer, new_indexer = ax._reindex_non_u
nique(keyarr)
   1551
-> 1552         self._validate_read_indexer(
   1553             keyarr, indexer, o._get_axis_number(axis), raise
_missing=raise_missing
   1554         )

~/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py in _
validate_read_indexer(self, key, indexer, axis, raise_missing)
   1638             if missing == len(indexer):
   1639                 axis_name = self.obj._get_axis_name(axis)
-> 1640                 raise KeyError(f"None of [{key}] are in the
 [{axis_name}]")
   1641
   1642             # We (temporarily) allow for some missing keys w
ith .loc, except in

KeyError: "None of [Index(['best_syn'], dtype='object')] are in the
 [index]"
```

In [43]:

```
data_bonus
```

Out[43]:

	suggestion_ger	synsets_ger	lexunits	hypernyms	lexunits_
0	aa	[Synset(id=s26358, lexunits=Kot, Scheiße, Exkr...	[Kot, Scheiße, Exkrement, Stuhl, Kacke, Kaka, ...	{Synset(id=s26356, lexunits=Fäkalien)}	
6	abbruch	[Synset(id=s106285, lexunits=Abbruch), Synset(...	[Abbruch, Abbruch, Beendigung, Beenden, Aufhör...	{Synset(id=s17129, lexunits=Schaden, Schädigung)}	
15	abnehmen	[Synset(id=s52508, lexunits=wegnehmen, abnehme...	[wegnehmen, abnehmen, fortnehmen, abchecken, a...	{Synset(id=s52497, lexunits=nehmen)}	
18	abschied	[Synset(id=s17481, lexunits=Abschied, Lebewohl...	[Abschied, Lebewohl, Verabschiedung, Abschied]	{Synset(id=s17478, lexunits=Trennung)}	
26	adel	[Synset(id=s32247, lexunits=Adelstitel, Adel, ...	[Adelstitel, Adel, Adelsbezeichnung, Adelsgesc...	{Synset(id=s32215, lexunits=Titel)}	
...	...	...	...	...	
3741	zirkus	[Synset(id=s108415, lexunits=Zirkus), Synset(i...	[Circus, Zirkus, Circus, Zirkus, Zirkusunterne...	{Synset(id=s42745, lexunits=Arena, Stadion)}	[Are
3742	zitat	[Synset(id=s32274, lexunits=Zitat), Synset(id=...	[Zitat, Zitat]	{Synset(id=s32267, lexunits=Redewendung, Idiom...	[Re Idiom
3752	zug	[Synset(id=s76189, lexunits=Zug), Synset(id=s4...	[Zug, Luftzug, Zug, Luft, Gesichtszug, Zug, Zu...	{Synset(id=s24150, lexunits=Formation, Gruppie...	
3753	zukunft	[Synset(id=s51054, lexunits=Zukunft, Vorzeitig...	[Zukunft, Vorzeitigkeit, Hinkunft, Futur, Zuku...	{Synset(id=s51051, lexunits=Zeitstufe)}	
3755	zusammenbruch	[Synset(id=s17538, lexunits=Kollaps, Zusammenb...	[Kollaps, Zusammenbruch, Zusammenbruch, Crash]	{Synset(id=s17096, lexunits=Vorfall, Begebenhe...	[Vorfall, E

451 rows × 11 columns

In [46]:

```
data_bonus.to_excel("data.xlsx")
```