Final project Data analysis for NLP

Manon Fleury
22410952

November 11, 2024

1 Analyzing CommonVoice

First we merge all files with the clip_durations.tsv file on "path" (corresponding to "clip").

```
base_path = '/content/drive/MyDrive/metadata'
languages = ["ar", "de", "en", "es", "fr", "id", "it", "ja", "lg", "ru", "sw", "ta", "tr", "zh-CN"]
all_data = [] # to store the DataFrams for each language

def merge_files(language):
    lang_path = os.path.join(base_path, language, 'cv-corpus-15.0-2023-09-08', language)
    clip_durations = pd.read_csv(os.path.join(lang_path, 'clip_durations.tsv'), sep='\t')
    files_to_merge = ['validated.tsv', 'train.tsv', 'dev.tsv', 'test.tsv', 'other.tsv', 'invalidated.tsv'] # without reported.tsv (doesnt have the "path" column)
    merged_lang_df = pd.DataFrame()

    for file_name in files_to_merge:
        file_path = os.path.join(lang_path, file_name)
        try:
            df_file = pd.read_csv(file_path, sep='\t')
            merged_df = pd.merge(df_file, clip_durations, left_on='path', right_on='clip', how='inner')
            merged_lang_df = pd.concat([merged_lang_df, merged_df], ignore_index=True)
        except pd.errors.ParserError as e: # to exclude one case (sw, other.tsv, row 306596 doesn't have correct format)
            print(f"Error of parsing for the file {file_name} in the language {language}: {e}")
    merged_lang_df.drop_duplicates(inplace=True)
    return merged_lang_df

for lang in languages:
    merged_lang_data = merge_files(lang)
    all_data.append(merged_lang_data)

df = pd.concat(all_data, ignore_index=True) # combine all dataframes of languages into one big dataframe
df.rename(columns={"locale": "language"}, inplace=True)

```

```
print(df.head())
```

Output : We end up with a dataframe with the following columns: client_id, path, sentence, up_votes, down_votes, age, gender, accents, variant, language, segment, clip, duration.

1. For each language, determine the total duration of recording and the number of different speakers.

```
summary_df = df.groupby('language').agg(
    total_duration=('duration[ms]', 'sum'),
    num_speakers=('client_id', 'nunique')
).reset_index()

summary_df['total_duration_hours'] = summary_df['total_duration'] / (1000 * 60 * 60) # convert from ms to hours

print(summary_df[['language', 'total_duration', 'total_duration_hours', 'num_speakers']])
```

Output :

Language	Total Duration (ms)	Total Duration (hours)	Number of Speakers
ar	553629078	153.785855	1481
de	4994673077	1387.409188	18352
en	12042063547	3345.017652	88901
es	7873294410	2187.026225	25338
fr	3902565718	1084.046033	17911
id	229541988	63.761663	516
it	1377995268	382.776463	6977
ja	797361045	221.489179	1713
lg	1883191947	523.108874	640
ru	932842296	259.122860	3053
sw	1797629004	499.341390	1149
ta	1357774244	377.159512	814
tr	408678399	113.521777	1511
zh-CN	3794298540	1053.971817	6823

2. For each language and each genre determine the total duration of recordings, the number of different speakers, the average, median, min and max number of recordings per speaker.

```
language_gender_stats = df.groupby(['language', 'gender']).agg(
    total_duration=('duration[ms]', 'sum'),
    unique_speakers=('client_id', 'nunique')
).reset_index()

recordings_per_speaker = df.groupby(['language', 'gender', 'client_id']).size().reset_index(name='recording_count')

recordings_stats = recordings_per_speaker.groupby(['language', 'gender']).agg(avg_recordings_per_speaker=('recording_count', 'mean'),
    median_recordings_per_speaker=('recording_count', 'median'),
    min_recordings_per_speaker=('recording_count', 'min'),
    max_recordings_per_speaker=('recording_count', 'max')).reset_index()
```

```
12
13 language_gender_stats = pd.merge(language_gender_stats, recordings_stats,
       on=['language', 'gender']) # merge the 2 df to have all the stats in
       the same table
14 language_gender_stats['total_duration_hours'] = language_gender_stats['
       total_duration'] / (1000 * 60 * 60) # ms to hours
15
16 print(language_gender_stats)
```

lang	gndr	tot_dur	uniq_spkr	avg_rec/spkr	med_rec/spkr	min_rec/spkr	max_rec/spkr	tot_hrs
ar	f	94 842 120	126	178.44	40.0	1	2200	26.35
ar	m	156 973 841	395	96.73	15.0	1	6177	43.60
ar	o	315 792	9	9.44	6.0	1	22	0.09
de	f	407 455 547	734	107.64	20.0	1	11 499	113.18
de	m	2 875 066 776	4143	135.95	25.0	1	64 421	798.63
de	o	32 227 155	62	104.98	20.0	1	2892	8.96
en	f	2 118 742 640	4689	82.48	15.0	1	25 559	588.54
en	m	5 285 065 589	17 995	58.05	12.0	1	60 115	1468.07
en	o	229 795 309	445	103.07	15.0	1	14 227	63.83
es	f	2 550 731 259	1687	308.29	15.0	1	37 041	708.54
es	m	4 269 296 518	4352	204.53	15.0	1	133 909	1185.92
es	o	18 143 340	99	35.39	10.0	1	453	5.04
fr	f	409 568 341	907	87.37	17.0	1	10 339	113.77
fr	m	2 273 612 769	3678	125.04	20.0	1	47 145	631.56
fr	o	30 139 740	85	73.81	30.0	2	999	8.38
id	f	62 317 680	43	356.35	25.0	3	4001	17.31
id	m	98 909 304	183	141.04	20.0	1	4019	27.47
id	o	9 766 260	4	508.00	130.5	10	1761	2.71
it	f	152 480 838	378	73.17	15.0	1	2796	42.36
it	m	810 947 993	1610	96.18	15.0	1	8915	225.82
it	o	5 729 388	15	68.73	10.0	5	687	1.59
ja	f	366 481 800	319	216.51	96.0	1	7484	101.80
ja	m	301 388 049	698	97.25	89.0	1	149	83.72
ja	o	3 443 196	16	49.88	28.0	4	11 809	0.96
lg	f	738 989 835	552.48	85.24	198.0	1	25 627	205.28
lg	m	668 571 096	497.31	71.22	100.0	1	5740	185.71
ru	f	145 245 908	90.07	57.82	10.0	1	7139	40.35
ru	m	553 533 251	108.29	66.63	15.0	3	200	153.76
ru	o	2 229 336	9	48.33	20.0	5	241	0.62
sw	f	526 278 240	324.58	56.89	48.0	1	6604	146.19
sw	m	659 990 736	382.68	82.93	35.0	1	30 213	183.33
sw	o	670 932	1	110.00	110.0	110	110	0.19
ta	f	156 643 088	231.72	45.23	30.0	1	7374	43.51
ta	m	178 394 263	110.32	59.86	13.0	1	8883	49.55
ta	o	335 448	2	32.50	32.5	6	86	0.09
tr	f	119 451 936	86	391.89	24.5	2	21 073	33.18
tr	m	174 381 213	449	110.17	15.0	1	13 079	48.44
tr	o	1 628 304	10	41.40	12.5	5	458	0.45
zh-CN	f	44 148 693	195	44.65	15.0	2	2581	12.27
zh-CN	m	233 751 490	937	49.04	10.0	1	2145	64.93
zh-CN	o	8 107 812	32	50.00	12.5	1	728	2.25

3. Which language has the highest proportion of recordings by a person in the largest age bracket considered ?

```
max_age_bracket = df['age'].dropna().unique().max()
age_bracket_df = df[df['age'] == max_age_bracket]
total_recordings_per_language = df.groupby('language').size().reset_index(
    name='total_recordings')
recordings_per_person = age_bracket_df.groupby(['language', 'client_id']).
    size().reset_index(name='recordings_by_person')
top_recordings_per_language = recordings_per_person.groupby('language').
    apply(lambda x: x.nlargest(1, 'recordings_by_person')).reset_index(drop
    =True)
merged_df = pd.merge(total_recordings_per_language,
    top_recordings_per_language, on='language', how='left')
merged_df['proportion_by_top_person'] = merged_df['recordings_by_person']
    / merged_df['total_recordings']
highest_proportion_language = merged_df.loc[merged_df['
    proportion_by_top_person'].idxmax()]
```

Output :
language: lg (Luganda)
proportion by top person: 0.079

4. Determine the number of different genres considered in the corpus. What is the percentage of recordings for which no genre information is available ? Plot the genre distribution (including cases where gender information is not available). Do the same for ages.

```
# df['gender'].unique() # ['male', nan, 'female', 'other']
df['gender'].nunique() # 3
```

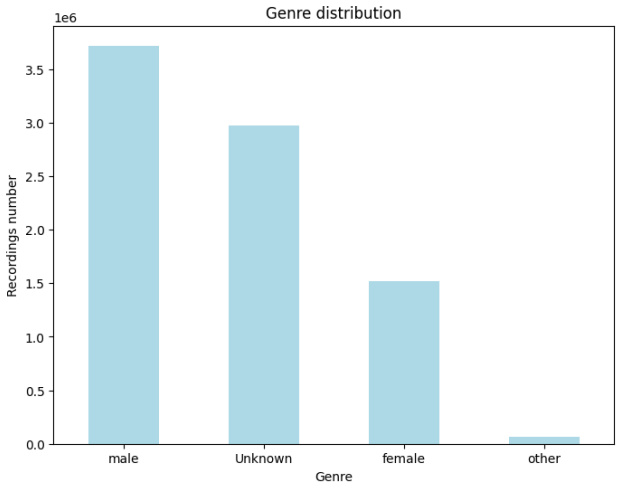
Output: We have 3 different genres.

```
(df['gender'].isna().sum() / len(df)) * 100
```

Output: 35.9% of recordings have no genre information available.

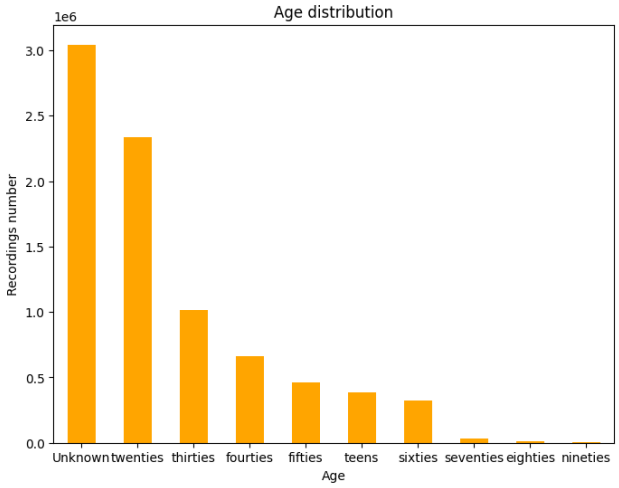
```
import matplotlib.pyplot as plt
df_gender = df.copy()
df_gender['gender'] = df_gender['gender'].fillna('Unknown')
genre_distr = df_gender['gender'].value_counts()
print(f"Genre pourcentage:{(genre_distr / genre_distr.sum()) * 100}")

plt.figure(figsize=(8, 8))
plt.pie(genre_distr, labels=genre_distr.index, autopct='%1.1f%%',
    startangle=90, colors=['lightblue', 'gray', 'lightgreen', 'blue'])
plt.title("Genre distribution")
plt.show()
```

Genre percentage:
male: 44.9%
Unknown: 35.9%
female: 18.3%
other: 0.9%



Same for ages:

```
df_age = df.copy()
df_age['age'] = df_age['age'].fillna('Unknown')
age_counts = df_age['age'].value_counts()

plt.figure(figsize=(8, 6))
age_counts.plot(kind='bar', color='orange')
plt.title("Age distribution")
plt.xlabel("Age")
plt.ylabel("Recordings number")
plt.xticks(rotation=0)
plt.show()
```



5. Find out for each language and each genre the 7 speakers who produced the most recordings.

```
f_dropna = df.dropna(subset=['language','gender'])
recordings_per_speaker = df_dropna.groupby(['language', 'gender', '
    client_id']).size().reset_index(name='recordings_count')
```

```
top_speakers = (
    recordings_per_speaker
    .groupby(['language', 'gender'])
    .apply(lambda x: x.nlargest(7, 'recordings_count'))
    .reset_index(drop=True)
)

top_speakers.head(20)
```

	language	gender	client_id	recordings_count
0	ar	female	78c954e30fd3a81e4abc72008b6c25427f6cb545275c59...	3435
1	ar	female	5f810213ca8e05e0d27a618d7a9e06c8ce4a2f4ca21eab...	3219
2	ar	female	1ebda36c79c7942d2d10a68236599dd5be279fcf6f17ad...	2068
3	ar	female	cf4d8f89b96eba7a58ae067df81ab438464c208f12b5f0...	1679
4	ar	female	d7517b67ab736aa3e93a6ff78544e9f6e5be2540651522...	1416
5	ar	female	fc3b87e39142b5fced5eb2422f0f5277bdf471c8865e9a...	1322
6	ar	female	6daa180b5a8f3703a86898bbacdda8bb2e72f7af448508...	1261
7	ar	male	d07077b6446f3b392e660d42eb32a30c4f6c363f9324e1...	6214
8	ar	male	1a3d0c01953a8e5095bddfd42ddad249e8f913610b8d3e...	2965
9	ar	male	9ffb4f97bb471ed746e6afd0e166bb8da6951fd612241b...	2214
10	ar	male	3ed0ada6e07c029a8e33c022429b902ecda77dbb159e19...	1568
11	ar	male	17072b6674e2589575c3c4009e0fef5e8c3a9e2ce92019...	1463
12	ar	male	4b4a821639d2905ef5a3097fdd9fea56ddb5114331c697...	1102
13	ar	male	3411d26e14d4532b4cb0d6aa9638c8c8136c2804620a55...	1098
14	ar	other	28115b2e4df8f98637d0192d79f49f510f4328a4b3c9b4...	41
15	ar	other	e98991e479da1b4ff4476aa5730834fa2a9e586d0cbd08...	25
16	ar	other	41bd311fae249fa6828bdf0598acd6b5f676e02c452c80...	12
17	ar	other	dc4e423628ce9a883b5ae0f997c40c6499f27fcade8d04...	10
18	ar	other	6846cf324073005ae481a6bb1fc38acc8195f8389d8b2c...	9
19	ar	other	1d3492a3acdc2d354e3eca0b55b4ab5665f5158aefdc58...	8

6. For each language, create a 1-hour test set and a training set as large as possible, ensuring that speakers from the test set do not appear in the training set.

```
import numpy as np

def create_set(df, language):
    train_set = pd.DataFrame()
    test_set = pd.DataFrame()

    lang_df = df[df['language'] == language]
    speaker_groups = lang_df.groupby('client_id')
    shuffled_speakers = np.random.permutation(list(speaker_groups.groups.
        keys()))

    total_duration = 0
    for speaker_id in shuffled_speakers:
        speaker_data = speaker_groups.get_group(speaker_id)
        speaker_duration = speaker_data['duration[ms]'].sum()

        if total_duration + speaker_duration <= 3600000:
            test_set = pd.concat([test_set, speaker_data], ignore_index=
                True)
```

```
            total_duration += speaker_duration
        else:
            train_set = pd.concat([train_set, speaker_data], ignore_index=
                True)

    return train_set, test_set

train_sets = {}
test_sets = {}

for lang in languages:
    train, test = create_set(df, lang)
    train_sets[lang] = train
    test_sets[lang] = test
```

7. How many languages have more female than male recordings?

```
gender_counts = df.groupby('language')['gender'].value_counts().unstack(
    fill_value=0)
print((gender_counts['female'] > gender_counts['male']).sum())
```

Output: 1

```
# to display which language and the gender counts:
gender_counts[gender_counts['female'] > gender_counts['male']] # lg :
    Luganda
```

Output: We have one language that has more female than male recordings (Luganda).

8. For each language, create a train set containing 1 hour of recordings and a 10-minute test set so that : i) train and test are gender-balanced ii) the train and test speakers are different.

```
def create_balanced_sets(df, language):
  lang_df = df[df['language'] == language]
  gender_groups = lang_df.groupby('gender')

  train_set = pd.DataFrame()
  test_set = pd.DataFrame()

  for gender in gender_groups.groups.keys():
    gender_df = gender_groups.get_group(gender)
    speaker_groups = gender_df.groupby('client_id')

    shuffled_speakers = np.random.permutation(list(speaker_groups.groups.
        keys()))

    test_duration = 0
    test_speakers = []

    for speaker in shuffled_speakers:
      speaker_data = speaker_groups.get_group(speaker)
      speaker_total_duration = speaker_data['duration[ms]'].sum()
```

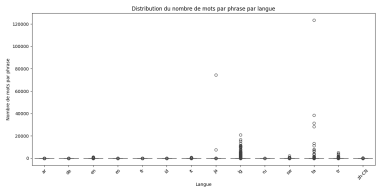
```
        if test_duration + speaker_total_duration <= 600000:  # 10 min in ms
            test_set = pd.concat([test_set, speaker_data], ignore_index=True)
            test_duration += speaker_total_duration
            test_speakers.append(speaker)

    train_duration = 0

    for speaker in shuffled_speakers:
        if speaker in test_speakers:
            continue  # to exclude the speakers from test

        speaker_data = speaker_groups.get_group(speaker)
        speaker_total_duration = speaker_data['duration[ms]'].sum()

        if train_duration + speaker_total_duration <= 3600000:  # 1 hour in
            ms
            train_set = pd.concat([train_set, speaker_data], ignore_index=True
                )
            train_duration += speaker_total_duration

    return train_set, test_set

train_sets = []
test_sets = []

for lang in languages:
    train_set, test_set = create_balanced_sets(df, lang)
    train_sets.append(train_set)
    test_sets.append(test_set)

final_train_set = pd.concat(train_sets, ignore_index=True)
final_test_set = pd.concat(test_sets, ignore_index=True)

print(f"Train set: {final_train_set.shape}")
print(f"Test set: {final_test_set.shape}")
```

Output:
Train set: (9997, 13)
Test set: (1655, 13)

9. Plot, for each language, a box plot representing the distribution of the number of words per sentence.

```
import seaborn as sns
import matplotlib.pyplot as plt
from polyglot.text import Text
from polyglot.detect import Detector

def count_words_polyglot(df,lang_hint):
    df_copy = df.copy()
    df_copy['word_count'] = df_copy['sentence'].apply(lambda x: len(Text(x
        , hint_language_code=lang_hint).words) if isinstance(x, str) else
        0)
```

```
    return df_copy

word_counts = []
for lang in languages:
    lang_df = df[df['language'] == lang]
    counted_df = count_words_polyglot(lang_df, lang_hint=lang)
    word_counts.append(counted_df[['language', 'word_count']])

all_word_counts = pd.concat(word_counts, ignore_index=True)

plt.figure(figsize=(12, 6))
sns.boxplot(x='language', y='word_count', data=all_word_counts)
plt.title('Distribution of the number of words per sentence per language')
plt.xlabel('Language')
plt.ylabel('Number of words per sentence')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



This graph is obviously wrong, and this could be explained by the \t separation in the files that poses a problem for the correct separation of sentences (that is why in this graph we see that the language 'ta' has more than 120,000 words per sentence).

10. For each language, find the 17 most frequent words after having removed stop words.

```
from polyglot.text import Text
from collections import Counter
from nltk.corpus import stopwords
import nltk

nltk.download('stopwords')

language_map = {'ar': 'arabic','de': 'german','en': 'english','es': '
    spanish','fr': 'french','id': 'indonesian','it': 'italian','ja': '
    japanese','lg': 'ganda','ru': 'russian','sw': 'swahili','ta': 'tamil','
    tr': 'turkish','zh-CN': 'chinese'}

def most_frequent_words(df, lang, top_n=17):
    stop_words = set(nltk.corpus.stopwords.words(language_map[lang])) if
        language_map[lang] in stopwords.fileids() else set()
    all_words = []
    for sentence in df['sentence']:
        if isinstance(sentence, str):
            tokens = [word.lower() for word in Text(sentence,
                hint_language_code=lang).words]
```

```
            filtered_words = [word for word in tokens if word not in
                stop_words and word.isalpha()]
            all_words.extend(filtered_words)

    word_counts = Counter(all_words)
    most_common_words = word_counts.most_common(top_n)
    return most_common_words

frequent_words_by_language = {}
for lang in df['language'].unique():
    lang_df = df[df['language'] == lang]
    frequent_words_by_language[lang] = most_frequent_words(lang_df, lang=
        lang, top_n=17)

for lang, words in frequent_words_by_language.items():
    print(f"Langue: {lang}")
    for word, count in words:
        print(f"{word}: {count}")
    print("\n")
```

Output:

```
Langue: ar
سامي: 4448
نوم: 2398
الله: 2284
ليلى: 1659
لقد: 1279
كانت: 1004
كنت: 989
يوم: 887
اليوم: 818
قال: 780
شيء: 752
فضلك: 718
أريد: 697
فاضل: 689
إنها: 641
أحب: 629
الناس: 605

Langue: de
wurde: 46321
wurden: 15382
zwei: 14510
gibt: 13527
heute: 12975
hauptstadt: 11984
mehr: 11345
schon: 10980
jedoch: 10583
zeit: 10170
drei: 9956
immer: 9156
liegt: 8870
mal: 8772
später: 8679
dabei: 8642
müssen: 8296

Langue: en
also: 96676
one: 67774
two: 54805
first: 39843
time: 34879
said: 33058
new: 31851
many: 31811
three: 30212
boy: 29570
would: 28182
used: 27347
like: 27073
school: 25313
later: 24774
know: 23147
people: 22041

Langue: es
dos: 45112
encuentra: 27684
tres: 25937
parte: 24326
ser: 23428
ciudad: 23144
además: 21408
años: 20199
actualmente: 20197
nombre: 19087
embargo: 18029
puede: 17403
año: 17342
gran: 16996
cuatro: 16698
primera: 16024
universidad: 15567

Langue: fr
a: 47211
cette: 29982
plus: 29338
deux: 29130
quatre: 17179
également: 17132
rue: 16796
cent: 16525
comme: 16102
trois: 16021
fait: 15217
aussi: 13902
vingt: 12268
alors: 12208
saint: 10793
être: 9926
cinq: 9714

Langue: id
tom: 3643
pergi: 1972
orang: 1782
rumah: 1302
bahasa: 1268
makan: 1257
jalan: 1220
buku: 1008
suka: 973
anak: 940
jepang: 814
belajar: 805
malam: 801
kemarin: 798
laki: 775
memiliki: 725
sekolah: 707

Langue: it
due: 8095
stato: 5974
dopo: 5694
parte: 5225
viene: 4903
essere: 4889
anni: 4513
prima: 4015
inoltre: 3737
venne: 3607
molto: 3566
tre: 3375
film: 3161
nome: 3126
città: 3088
poi: 3038
solo: 2883

Langue: ja
の: 96330
は: 73902
に: 69306
が: 60710
を: 56005
た: 41490
で: 39207
て: 34382
的: 34341
と: 34304
ない: 26818
ある: 25201
な: 24418
も: 23679
って: 21775
っ: 19351
こと: 18299

Langue: lg
mu: 104221
ku: 64282
nga: 43749
lg: 31098
nti: 25758
ne: 22113
abantu: 21441
nnyo: 18630
era: 16205
bwe: 15057
buli: 11677
female: 10694
male: 10531
naye: 10127
twenties: 9171
ye: 8548
ki: 8509

Langue: ru
это: 13727
также: 4454
слово: 3521
должны: 3411
является: 3154
нам: 3021
безопасности: 2713
наций: 2371
объединенных: 2281
очень: 2211
имеет: 2200
организации: 2192
поэтому: 2160
время: 2060
конференции: 2055
сегодня: 2016
однако: 1935

Langue: sw
ya: 156931
na: 148208
wa: 104559
kwa: 56972
ni: 45715
katika: 35312
za: 31781
la: 30574
moja: 17577
kuwa: 16972
cha: 16394
kama: 16261
tanzania: 12827
sana: 12702
watu: 12326
elfu: 12149
kwenye: 12006

Langue: ta
ta: 23140
female: 7553
male: 6994
twenties: 6733
thirties: 4011
படம்: 3316
fourties: 2445
fifties: 1095
என: 968
உலக: 376
வர: 365
teens: 289
வட: 245
ஆக: 217
tamil: 189
nadu: 189
சமய: 170

Langue: tr
bir: 14884
mi: 2741
var: 2716
kadar: 2582
ben: 2530
değil: 2467
iki: 2107
sonra: 1944
tr: 1842
fakat: 1772
bunu: 1678
yok: 1599
sen: 1492
bana: 1427
beni: 1416
on: 1413
male: 1363

Langue: zh-CN
一个: 92267
科: 56137
县: 39909
尔: 38056
种: 31938
属下: 30750
州: 30504
斯: 30229
属: 26532
人: 26327
位于: 26313
区: 25733
中: 25321
会: 21078
图: 21070
美国: 20975
中国: 20222
```

11. Compute, for each language, the size of the available data both in term of duration and in terms of number of words.

```
language_size_stats = df.groupby('language').agg(
    total_duration=('duration[ms]', 'sum'),
    total_words=('sentence', lambda x: x.str.split().str.len().sum())
).reset_index()

```

```
language_size_stats['total_duration_hours'] = language_size_stats['
    total_duration'] / (1000 * 60 * 60)

print(language_size_stats[['language', 'total_duration', '
    total_duration_hours', 'total_words']])
```

Output:

	language	total_duration	total_duration_hours	total_words
0	ar	553629078	153.785855	738550.0
1	de	4994673077	1387.409188	8460379.0
2	en	12042063547	3345.017652	21901656.0
3	es	7873294410	2187.026225	15541815.0
4	fr	3902565718	1084.046033	7333271.0
5	id	229541988	63.761663	359489.0
6	it	1377995268	382.776463	2533228.0
7	ja	797361045	221.489179	195373.0
8	lg	1883191947	523.108874	2862555.0
9	ru	932842296	259.122860	1544343.0
10	sw	1797629004	499.341390	3006125.0
11	ta	1357774244	377.159512	1864695.0
12	tr	408678399	113.521777	538442.0
13	zh-CN	3794298540	1053.971817	834422.0

2 Assessing the variability of wav2vec2 representations

12. Explain each instruction of the manatee function.

```
def manatee(cow, yak, pelican, slug):
    cow = cow.to_frame().T.reset_index().copy()
```

Converts cow (likely a pandas Series) to a dataframe, transposes it so that rows become columns and vice versa, resets the index to a default integer index and creates a copy of the dataframe.

```
butterfly = cow[[yak]]
  .explode(column=yak)
  .apply(lambda x: {"mussel": x[yak].minTime,
                    "end_time": x[yak].maxTime,
                    "annotation": x[yak].mark,
                    },
         axis=1,
         result_type="expand",
         )
```

Selects the yak column (which holds word-level annotations), expands lists of annotations in the yak column into separate rows, applies a function to each row which extracts information from the word annotations: the start time of the word (minTime), the end time (maxTime), the actual annotation/label for the word. Then it ensures the output is expanded into separate columns. The re

```
crane_fly = cow[[pelican]].explode(column=pelican)
```

Expands the pelican column from cow (list of representations for a layer) into separate rows, creating individual word representations for each instance

```
crane_fly["leaf_barnacle"] = slug
crane_fly["leaf_barnacle"] = crane_fly["leaf_barnacle"].cumsum() - slug
```

Adds a new column leaf_barnacle to crane_fly, initially setting all values to slug (1/49, time step between frames).
Then, leaf_barnacle is modified by calculating the cumulative sum of the previous values, and subtracting slug from the result. This likely normalizes the time step across representations.

```
crane_fly["speaker"] = cow["speaker"]
crane_fly["filename"] = cow["filename"]
crane_fly["sentence"] = cow["sentence"]
```

Adds the speaker, filename and sentence columns from cow to crane_fly.

```
return pd.merge_asof(
    crane_fly, butterfly, left_on="leaf_barnacle", right_on="mussel"
)
```

"as-of" merge, ie matching rows from the crane_fly (the word representations) with rows from butterfly (the word annotations) based on their time or frame-related columns, ie matching the leaf_barnacle column (from crane_fly) to the mussel column (from butterfly).
This aligns the word-level annotations (start and end times, and labels) with their corresponding word representations.

13. Using the manatee function, create a data frame that, for each utterance of a word in the corpus, includes : the word, the speaker and their gender, and a matrix corresponding to the representation on the last layer of this utterance predicted by wav2vec2. This matrix should have dimensions of number of frames $\times$ 1,024.

```
import numpy as np
from tqdm import tqdm

def manatee(cow, yak, pelican, slug):
  cow = cow.to_frame().T.reset_index().copy()

  butterfly = cow[[yak]].explode(column=yak).apply(lambda x: {"mussel": x[
      yak].minTime,
                    "end_time": x[yak].maxTime,
                    "annotation": x[yak].mark,
                    },
          axis=1,
          result_type="expand",
          )
  crane_fly = cow[[pelican]].explode(column=pelican)
  crane_fly["leaf_barnacle"] = slug
  crane_fly["leaf_barnacle"] = crane_fly["leaf_barnacle"].cumsum() - slug

  crane_fly["speaker"] = cow["speaker"]
  crane_fly["filename"] = cow["filename"]
  crane_fly["sentence"] = cow["sentence"]

  return pd.merge_asof(
      crane_fly, butterfly, left_on="leaf_barnacle", right_on="mussel"
      )


df2 = pd.read_pickle(file_path2) # containing exo2_small.pkl

vect_df = pd.DataFrame(columns=['word', 'speaker', 'gender'])

for column, row in tqdm(df2.iterrows(), total=df2.shape[0]):
    df = manatee(
```

```
            row,
            yak="words",
            pelican='layer_24',
            slug=1 / 49,
    )

    df = df.rename(columns={'layer_24': "rpt"})

    temp_df = pd.DataFrame(columns=['word', 'speaker', 'gender'])
    temp_df['word'] = df['annotation'].unique()
    temp_df["speaker"] = df['speaker'].iloc[0]
    temp_df['gender'] = row['gender']

    annotation = df.groupby(['annotation'])['rpt'].agg(np.vstack).to_frame
        ()

    temp_df = pd.merge(temp_df, annotation, left_on='word', right_index=
        True)

    vect_df = pd.concat([vect_df, temp_df], ignore_index=True, axis=0)

vect_df.to_pickle('vector_df.pkl')
vect_df
```

Output:

	word	speaker	gender	rpt
0		DD	f	[[-0.039639957, 0.1330243, 0.1184451, -0.03256...
1	dis	DD	f	[[-0.035875946, 0.22794272, 0.0068324027, -0.0...
2	garage	DD	f	[[-0.033193443, 0.22977483, 0.0049105175, -0.0...
3	trois	DD	f	[[-0.03695189, 0.22511744, 0.007941497, -0.011...
4	fois	DD	f	[[-0.029429456, 0.23391968, 0.0060470877, -0.0...

14. Compute the DTW distances between all pairs of utterances of the same word. A function to compute this distance can be downloaded from the lecture page. To compute this distance, for each word, create all pairs of representations, compute the cosine distance between the frames of these two representations (i.e. if the two representations are of size n $\times$ 1,024 and m $\times$ 1,024, the resulting distance matrix will have a size of n $\times$ m), and then compute the DTW from the resulting distance matrix.

```
import numpy as np
from scipy.spatial.distance import cosine

def dtw(dist_mat):
    """
    Find minimum-cost path through matrix `dist_mat` using dynamic
        programming.

    The cost of a path is defined as the sum of the matrix entries on that
    path. See the following for details of the algorithm:

```

```
    - http://en.wikipedia.org/wiki/Dynamic_time_warping
    - https://www.ee.columbia.edu/~dpwe/resources/matlab/dtw/dp.m

    The notation in the first reference was followed, while Dan Ellis's
       code
    (second reference) was used to check for correctness. Returns a list
       of
    path indices and the cost matrix.
    """

    N, M = dist_mat.shape

    # Initialize the cost matrix
    cost_mat = np.zeros((N + 1, M + 1))
    for i in range(1, N + 1):
        cost_mat[i, 0] = np.inf
    for i in range(1, M + 1):
        cost_mat[0, i] = np.inf

    # Fill the cost matrix while keeping traceback information
    traceback_mat = np.zeros((N, M))
    for i in range(N):
        for j in range(M):
            penalty = [
                cost_mat[i, j],      # match (0)
                cost_mat[i, j + 1],  # insertion (1)
                cost_mat[i + 1, j]]  # deletion (2)
            i_penalty = np.argmin(penalty)
            cost_mat[i + 1, j + 1] = dist_mat[i, j] + penalty[i_penalty]
            traceback_mat[i, j] = i_penalty

    # Traceback from bottom right
    i = N - 1
    j = M - 1
    path = [(i, j)]
    while i > 0 or j > 0:
        tb_type = traceback_mat[i, j]
        if tb_type == 0:
            # Match
            i = i - 1
            j = j - 1
        elif tb_type == 1:
            # Insertion
            i = i - 1
        elif tb_type == 2:
            # Deletion
            j = j - 1
        path.append((i, j))

    # Strip infinity edges from cost_mat before returning
    cost_mat = cost_mat[1:, 1:]
    return {"path": path[::-1],
            "cost_mat": cost_mat,
            "cost": cost_mat[-1, -1]}
```

```python
def compute_dtw_for_word(df, word_column, layer_column):
    """Returns: dict: Dictionary with DTW distances between all pairs of
       utterances of the same word."""

    dtw_results = {}

    for word in df[word_column].unique():
        word_data = df[df[word_column] == word]
        representations = []
        for _, row in word_data.iterrows():
            representations.append(row[layer_column])

        n = len(representations)
        dist_mat = np.zeros((n, n))

        for i in range(n):
            for j in range(i, n):
                rep_i = representations[i]
                rep_j = representations[j]

                distance = np.zeros((rep_i.shape[0], rep_j.shape[0]))  # (
                   n_frames_i x n_frames_j)
                for f_i in range(rep_i.shape[0]):
                    for f_j in range(rep_j.shape[0]):
                        distance[f_i, f_j] = cosine(rep_i[f_i], rep_j[f_j
                           ])

                dtw_results[(word, i, j)] = dtw(distance)["cost"]

    return dtw_results


dtw_distances = compute_dtw_for_word(vect_df, word_column='word',
    layer_column='rpt')
```

The dtw provided function is too long to compute (> 4 hours), so results were not obtained.

15. Plot the distribution of distances for each word, distinguishing between cases where the two utterances were spoken by the same person and those where the speakers were different. What can you conclude ? Figure 2 gives you an overview of the plot you should generate.

If the dtw function would have been executed, we would have obtain something like this:

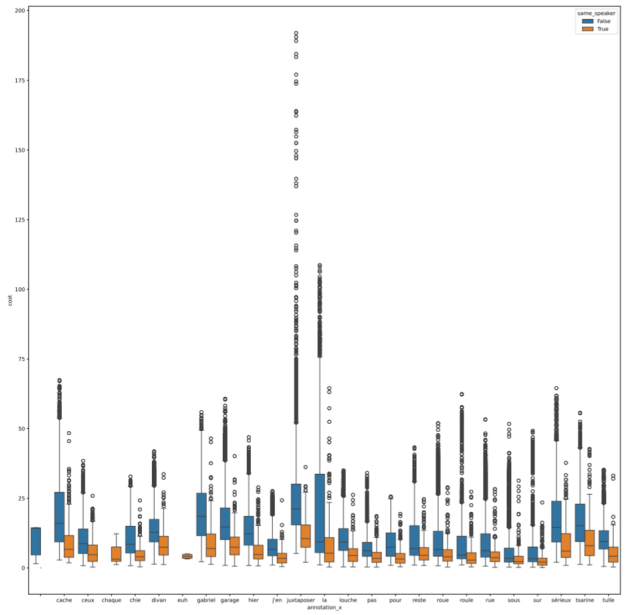


Figure 1: Distribution of distances for each word

We see the different words spoken by various speakers on the x-axis, and the cosine similarity spoken by the same speaker (True) versus different speakers (False) on the y-axis.
This plot analyzes how the wav2vec2 model consistently represents the same word spoken by the same speaker compared to different speakers. If the distances for the same speaker category are generally lower, it implies that the model captures consistent features across repeated utterances by the same speaker. Conversely, higher distances for different speaker instances would indicate inter-speaker variability, showing how different speakers produce distinct neural representations for the same word.