EDA

March 24, 2022

1 Eksploracja danych

[1]:
```
! python -m spacy download en_core_web_sm
```

```
Collecting en_core_web_sm==2.2.5
  Downloading https://github.com/explosion/spacy-
models/releases/download/en_core_web_sm-2.2.5/en_core_web_sm-2.2.5.tar.gz (12.0
MB)
     |                  | 12.0 MB 13.8 MB/s
Requirement already satisfied: spacy>=2.2.2 in
/usr/local/lib/python3.7/dist-packages (from en_core_web_sm==2.2.5) (2.2.4)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (2.23.0)
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.7/dist-
packages (from spacy>=2.2.2->en_core_web_sm==2.2.5) (1.21.5)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (4.63.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-
packages (from spacy>=2.2.2->en_core_web_sm==2.2.5) (57.4.0)
Requirement already satisfied: thinc==7.4.0 in /usr/local/lib/python3.7/dist-
packages (from spacy>=2.2.2->en_core_web_sm==2.2.5) (7.4.0)
Requirement already satisfied: wasabi<1.1.0,>=0.4.0 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (0.9.0)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (3.0.6)
Requirement already satisfied: catalogue<1.1.0,>=0.0.7 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (1.0.0)
Requirement already satisfied: plac<1.2.0,>=0.9.6 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (1.1.3)
Requirement already satisfied: srsly<1.1.0,>=1.0.2 in
/usr/local/lib/python3.7/dist-packages (from
```

```
spacy>=2.2.2->en_core_web_sm==2.2.5) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (1.0.6)
Requirement already satisfied: blis<0.5.0,>=0.4.0 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (0.4.1)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in
/usr/local/lib/python3.7/dist-packages (from
spacy>=2.2.2->en_core_web_sm==2.2.5) (2.0.6)
Requirement already satisfied: importlib-metadata>=0.20 in
/usr/local/lib/python3.7/dist-packages (from
catalogue<1.1.0,>=0.0.7->spacy>=2.2.2->en_core_web_sm==2.2.5) (4.11.3)
Requirement already satisfied: typing-extensions>=3.6.4 in
/usr/local/lib/python3.7/dist-packages (from importlib-
metadata>=0.20->catalogue<1.1.0,>=0.0.7->spacy>=2.2.2->en_core_web_sm==2.2.5)
(3.10.0.2)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-
packages (from importlib-
metadata>=0.20->catalogue<1.1.0,>=0.0.7->spacy>=2.2.2->en_core_web_sm==2.2.5)
(3.7.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests<3.0.0,>=2.13.0->spacy>=2.2.2->en_core_web_sm==2.2.5)
(2.10)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from
requests<3.0.0,>=2.13.0->spacy>=2.2.2->en_core_web_sm==2.2.5) (2021.10.8)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from
requests<3.0.0,>=2.13.0->spacy>=2.2.2->en_core_web_sm==2.2.5) (1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from
requests<3.0.0,>=2.13.0->spacy>=2.2.2->en_core_web_sm==2.2.5) (3.0.4)
 Download and installation successful
You can now load the model via spacy.load('en_core_web_sm')
```

[3]:
```
import spacy
import pandas as pd
from tqdm.auto import tqdm
import plotly.express as px
from wordcloud import WordCloud
from matplotlib import pyplot as plt
from datetime import datetime, timedelta
import numpy as np
import kaleido
from collections import Counter
import itertools
```

```
from spacy import displacy
import textstat
from textstat import flesch_reading_ease, difficult_words,␣
 ↪automated_readability_index, text_standard
import seaborn as sns
from nltk import ngrams

pd.options.plotting.backend = "plotly"
```

2 Data Loading

Data from https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/VGYI0E

[4]:
```
!wget -O data.tar.gz https://dataverse.harvard.edu/api/access/datafile/:
 ↪persistentId?persistentId=doi:10.7910/DVN/6MZN76/CRUNF0
```

```
--2022-03-24 21:05:22--  https://dataverse.harvard.edu/api/access/datafile/:pers
istentId?persistentId=doi:10.7910/DVN/6MZN76/CRUNF0
Resolving dataverse.harvard.edu (dataverse.harvard.edu)… 35.168.250.191,
35.170.94.248, 34.237.225.150
Connecting to dataverse.harvard.edu
(dataverse.harvard.edu)|35.168.250.191|:443… connected.
HTTP request sent, awaiting response… 303 See Other
Location: https://dvn-
cloud.s3.amazonaws.com/10.7910/DVN/6MZN76/15de5b930dd-0fada3dbae00?response-
content-disposition=attachment%3B%20filename%2A%3DUTF-8%27%27Dail_debates_1919-2
013.tar.gz&response-content-type=application%2Fx-gzip&X-Amz-Algorithm=AWS4-HMAC-
SHA256&X-Amz-Date=20220324T210522Z&X-Amz-SignedHeaders=host&X-Amz-
Expires=3600&X-Amz-Credential=AKIAIEJ3NV7UYCSRJC7A%2F20220324%2Fus-
east-1%2Fs3%2Faws4_request&X-Amz-
Signature=00b9d2e63f0da8267785f744d6cee7b9917ee45116500f791b549ec820f9c571
[following]
--2022-03-24 21:05:22--  https://dvn-
cloud.s3.amazonaws.com/10.7910/DVN/6MZN76/15de5b930dd-0fada3dbae00?response-
content-disposition=attachment%3B%20filename%2A%3DUTF-8%27%27Dail_debates_1919-2
013.tar.gz&response-content-type=application%2Fx-gzip&X-Amz-Algorithm=AWS4-HMAC-
SHA256&X-Amz-Date=20220324T210522Z&X-Amz-SignedHeaders=host&X-Amz-
Expires=3600&X-Amz-Credential=AKIAIEJ3NV7UYCSRJC7A%2F20220324%2Fus-
east-1%2Fs3%2Faws4_request&X-Amz-
Signature=00b9d2e63f0da8267785f744d6cee7b9917ee45116500f791b549ec820f9c571
Resolving dvn-cloud.s3.amazonaws.com (dvn-cloud.s3.amazonaws.com)…
52.217.74.241
Connecting to dvn-cloud.s3.amazonaws.com (dvn-
cloud.s3.amazonaws.com)|52.217.74.241|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 959382206 (915M) [application/x-gzip]
```

```
Saving to: ‘data.tar.gz’

data.tar.gz         100%[===================>] 914.94M  39.7MB/s    in 22s

2022-03-24 21:05:44 (42.2 MB/s) - ‘data.tar.gz’ saved [959382206/959382206]
```

[5]:
```
!tar -xf data.tar.gz
```

[6]:
```
en = spacy.load("en_core_web_sm") # loading spacy model
df = pd.read_table('Dail_debates_1919-2013.tab')
```

Dail_debates_1919-2013.tab - Ramka danych zawierająca dodatkowe informacje w kolumnach: -position
-department
-start_day
-end_day

[7]:
```
add_df = pd.read_table('Dail_debates_1937-2011_ministers.tab')
```

[8]:
```
add_df = add_df.loc[add_df.start_year > 2005] #wybieramy podzbiór od 2005r wzwyż
```

[9]:
```
add_df.start_date = pd.to_datetime(add_df.start_date)
add_df.end_date = pd.to_datetime(add_df.end_date)
```

2.0.1 Preprocesing

Wybieramy podzbiór danych od 2005r.

[10]:
```
df.date = pd.to_datetime(df.date)
df2 = df[df.date.dt.year>2005]
```

[11]:
```
# próbka danych
df2 = df2.sample(frac=.01, random_state=123)
len(df2)
```

[11]:
```
8198
```

[12]:
```
df2.isnull().any()
```

[12]:
```
speechID      False
memberID      False
partyID       False
constID       False
title         False
date          False
```

```
member_name    False
party_name     False
const_name     False
speech         False
dtype: bool
```

Ramka danych `df2` nie zawiera braków danych. Z kolei dodatkowa ramka danych ma braki jedynie w dacie zakończenia kadencji (analizując datę rozpoczęcia kadencji dla tych rekordów wywnioskowaliśmy, że brak danych oznacza trwającą kadencję).

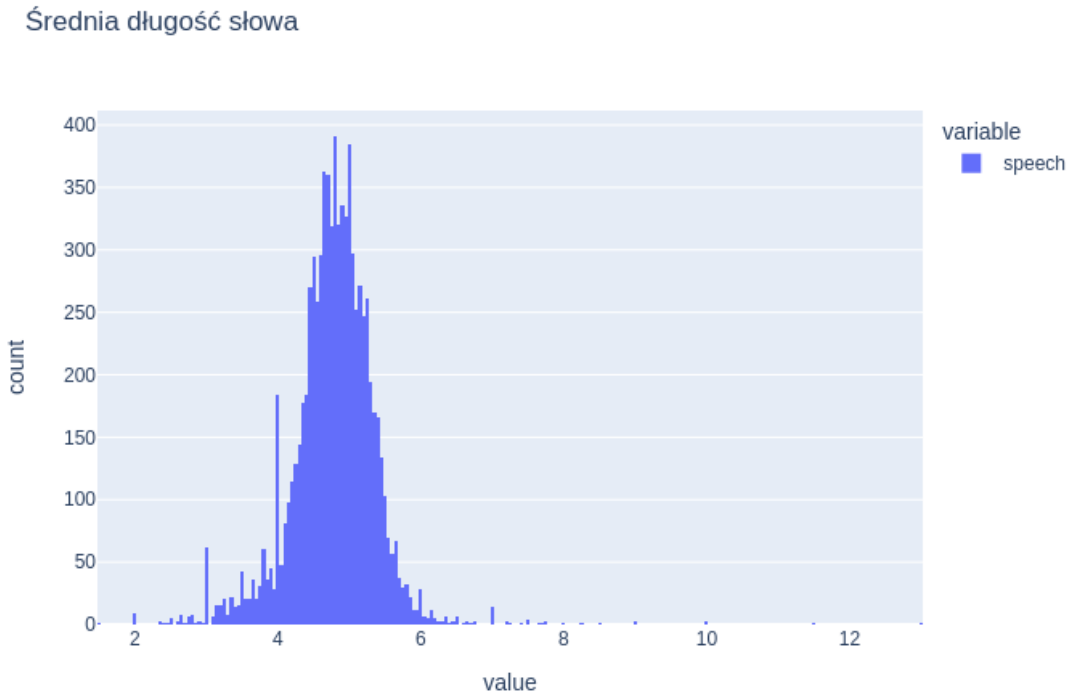
2.1 Statystyki ogólne

[13]:
```
a = df2.speech.apply(lambda x: textstat.sentence_count(x))
fig = px.histogram(a, title = 'Rozkład liczby zdań', log_y = True)
img_bytes = fig.to_image(format="png")
from IPython.display import Image
Image(img_bytes)
```

```
<Figure size 432x288 with 0 Axes>
```

[16]:
```
a = df2.speech.apply(lambda x: textstat.letter_count(x, ignore_spaces=True) /␣
 ↪textstat.lexicon_count(x, removepunct=True)) #średnia długość
fig = px.histogram(a, title = 'Średnia długość słowa')
img_bytes = fig.to_image(format="png")
from IPython.display import Image
Image(img_bytes)
```

[16]:



[15]:
```
print("Średnia długość słowa wynosi:", np.round(a.mean(), 2))
```

```
Średnia długość słowa wynosi: 4.77
```

```
Rozkład długości słów jest zbliżony do rozkładu normalnego.
```

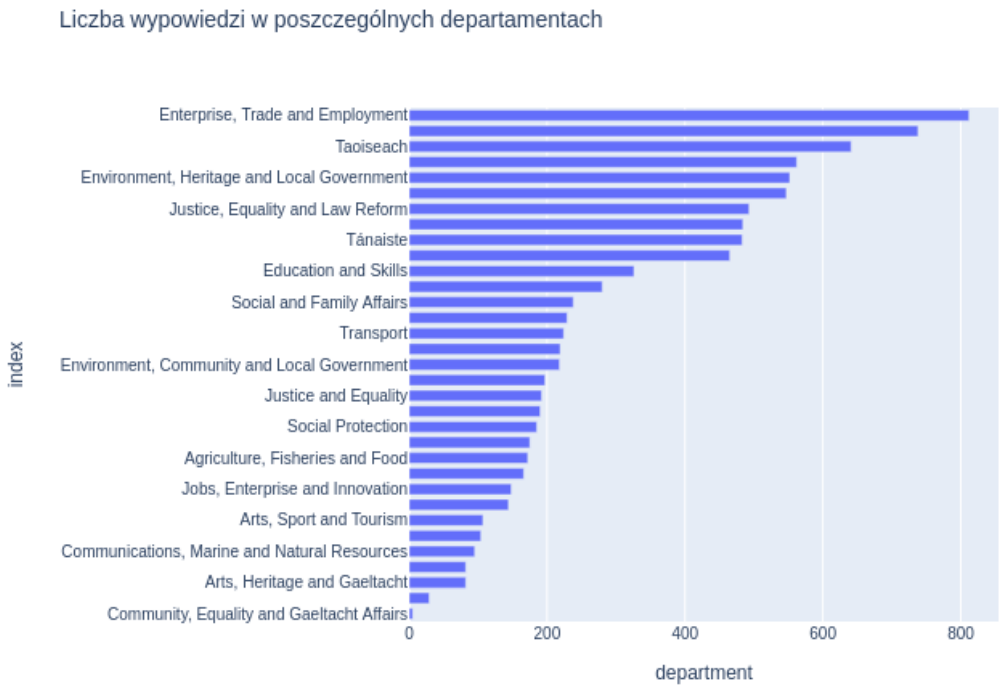
```
Łączymy ramki danych df2 oraz add_df jako merged_df
```

[17]:
```
merged_df = df2.merge(add_df, on = 'memberID', how = 'left')
```

2.2 Analiza wypowiedzi z podziałem na ministerstwa

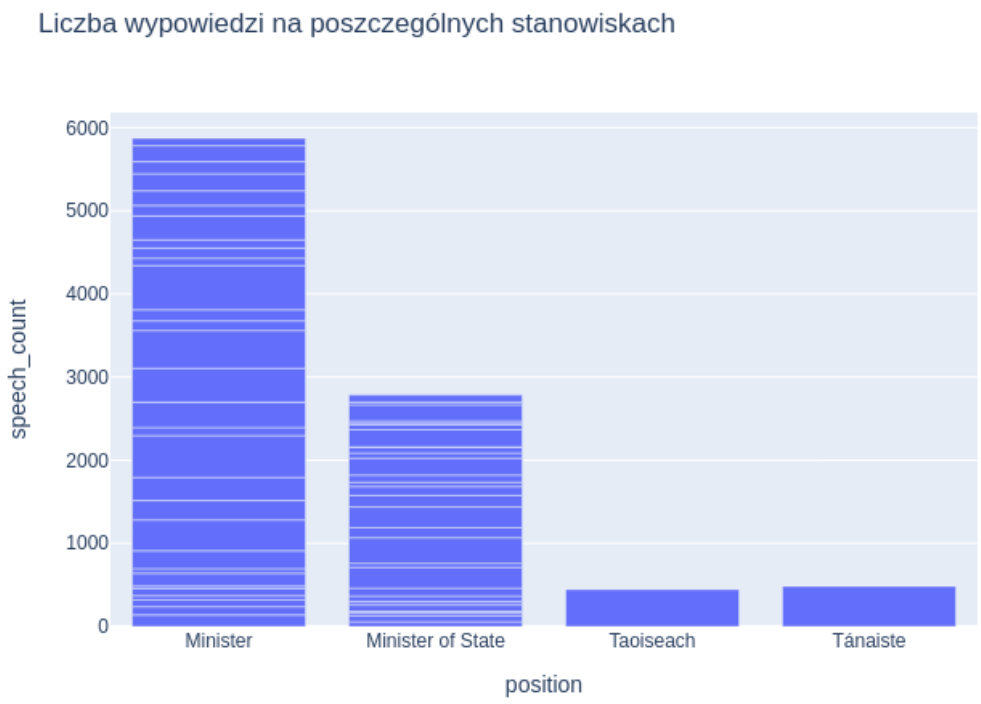
[18]:
```
dep_stat = pd.DataFrame(merged_df.department.value_counts()).reset_index()

fig = px.bar(dep_stat,orientation='h', y='index', x='department', title =␣
↪'Liczba wypowiedzi w poszczególnych departamentach')

fig['layout']['yaxis']['autorange'] = "reversed"
fig.update_layout(bargap=0.30, font={'size':10})
img_bytes = fig.to_image(format="png")
from IPython.display import Image
Image(img_bytes)
```

[18]:



[19]:
```
pos_dep_stat = merged_df.groupby(['position', 'department']).count().
↪reset_index()
pos_dep_stat = pos_dep_stat.iloc[:, 0:3] #SpeechID
pos_dep_stat = pos_dep_stat.rename(columns = {'speechID':'speech_count'})
fig = px.bar(pos_dep_stat, x = 'position', y = 'speech_count', title = 'Liczba␣
↪wypowiedzi na poszczególnych stanowiskach')
img_bytes = fig.to_image(format="png")
from IPython.display import Image
Image(img_bytes)
```

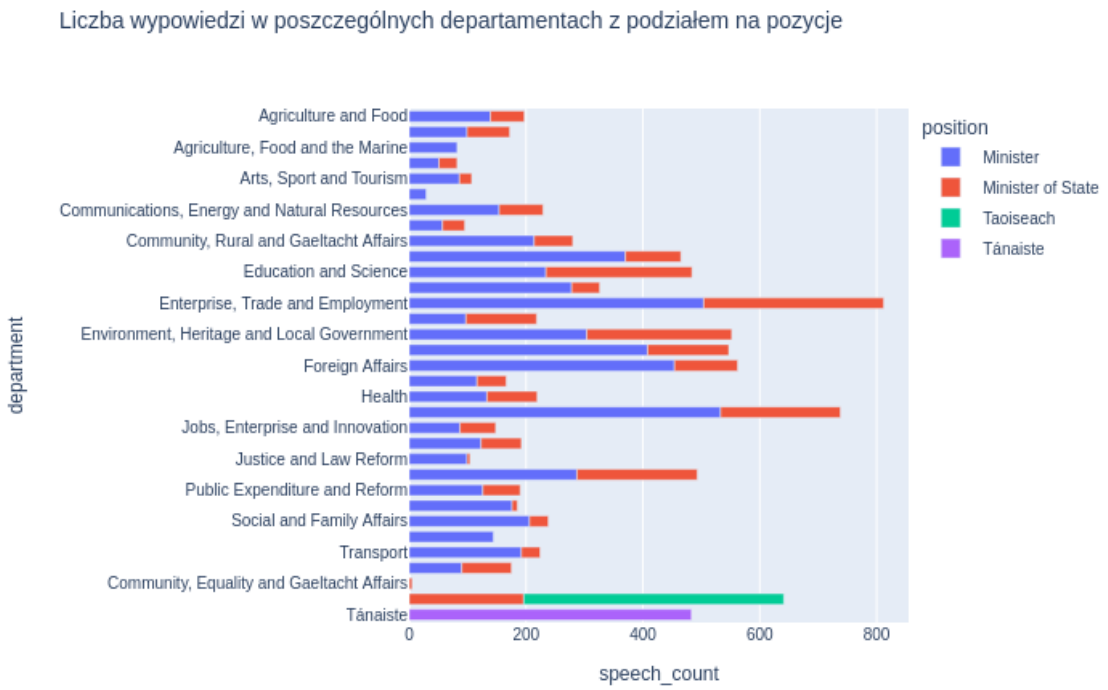
[19]:



W zbiorze dominują wypowiedzi ministrów. **Minister of state** to stanowisko asystenta ministra (inna nazwa to *Junior Minister*).
Taoiseach to premier Irlandii, a **Tánaiste** - zastępca premiera.

[20]:
```
fig = px.bar(pos_dep_stat,orientation='h', y='department', x='speech_count',␣
↪color = 'position', title = 'Liczba wypowiedzi w poszczególnych␣
↪departamentach z podziałem na pozycje')

fig['layout']['yaxis']['autorange'] = "reversed"
fig.update_layout(bargap=0.30, font={'size':10})
img_bytes = fig.to_image(format="png")
from IPython.display import Image
Image(img_bytes)
```

[20]:



2.3 Eksploracja z podziałem na 6 najpopularniejszych ministerstw

[21]:
```
popular_departments = list(merged_df.department.value_counts().index[0:6]) #na
↪razie weźmy top 6, potem można połączyć w obszary tematyczna
```

[22]:
```
top_6_departments = merged_df.loc[merged_df.department.
↪isin(popular_departments)]
```

[23]:
```
def tokenize_lemmatize(df_speech, docs_present = False): #funkcja zwraca listę
↪tokenów, opcjonalnie również dokumentów bez tokenizacji
  speeches_docs = []
  speeches = [] #tokenizacja i lematyzacja
  for speech in df_speech:
    doc = en(speech)
    tk = [token.lemma_ for token in doc if not token.is_stop if not token.
↪is_punct]
    speeches.append(tk)
    if docs_present:
      speeches_docs.append(doc)
  if docs_present:
```

```
    return speeches, speeches_docs
  return speeches
```

[24]:
```
[docs_dep_speeches_all,docs_speeches] = tokenize_lemmatize(merged_df.speech,␣
↪True)
```

[25]:
```
docs_dep_speeches = list(np.array(docs_dep_speeches_all)[top_6_departments.
↪index])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1:
VisibleDeprecationWarning:

Creating an ndarray from ragged nested sequences (which is a list-or-tuple of
lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If
you meant to do this, you must specify 'dtype=object' when creating the ndarray.
```

Najczęściej występujące słowa we wszystkich dokumentach (dla top 6 departamentów) Aż 4/10 słów jest bezpośrednio powiązanych z polityką.

[26]:
```
word_counts = Counter(list(itertools.chain(*docs_dep_speeches)))
word_counts.most_common(10)
```

[26]:
```
[('Department', 2248),
 ('Deputy', 2002),
 ('year', 1764),
 ('service', 1699),
 ('issue', 1568),
 ('provide', 1510),
 ('matter', 1373),
 ('Government', 1271),
 ('Minister', 1268),
 ('school', 1260)]
```

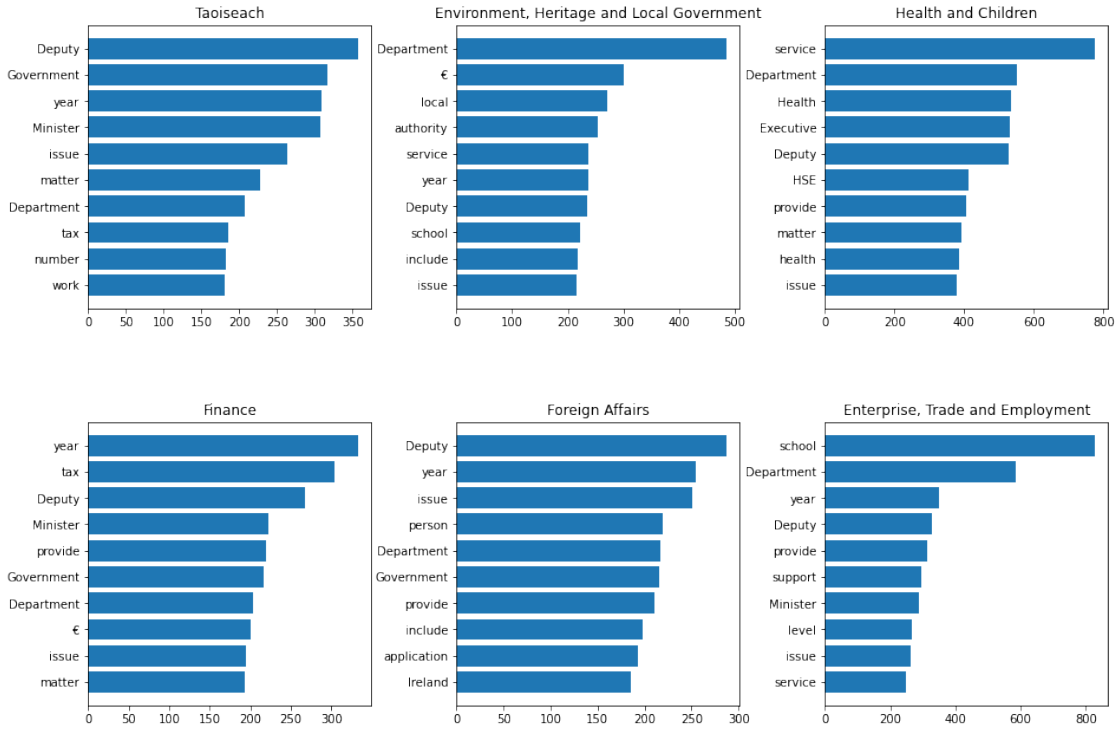
Słowa takie jak 'Department', 'Deputy', itp. pojawiają się bardzo często - możemy rozważyć potraktowanie ich jako stopwords.

[27]:
```
def divide_by_departments(docs_speeches, departments_names):
  docs_by_deps = []
  for dep in departments_names:
    pos = np.where(np.isin(top_6_departments.department, dep))
    arr = np.array(docs_speeches)[pos]
    arr = arr.tolist()
    merged = list(itertools.chain(*arr))
    docs_by_deps.append(merged)
  return docs_by_deps
```

[28]:
```
departments_names = pd.unique(top_6_departments.department)
docs_by_deps = divide_by_departments(docs_dep_speeches, departments_names)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5:
VisibleDeprecationWarning:

Creating an ndarray from ragged nested sequences (which is a list-or-tuple of
lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If
you meant to do this, you must specify 'dtype=object' when creating the ndarray.
```

[28]:
```
fig, ax = plt.subplots(2, 3)
fig.set_size_inches(15, 10)
a = 0
n_grams_list = []

for i in range(2):
  for j in range(3):
    word_counts = Counter(docs_by_deps[a])
    n_grams = ngrams(docs_by_deps[a], 3)
    n_grams_counter = Counter(n_grams).most_common(20)
    n_grams_list.append(n_grams_counter)
    counts = pd.DataFrame(word_counts.most_common(10), columns=['word',␣
 ↪'count']).sort_values('count')
    ax[i, j].barh(counts.iloc[:,0], counts.iloc[:, 1])
    ax[i, j].set_title(departments_names[a])
    a = a+1

    plt.subplots_adjust(left=0.1,
                        bottom=0.1,
                        right=0.9,
                        top=0.9,
                        wspace=0.3,
                        hspace=0.4)
```

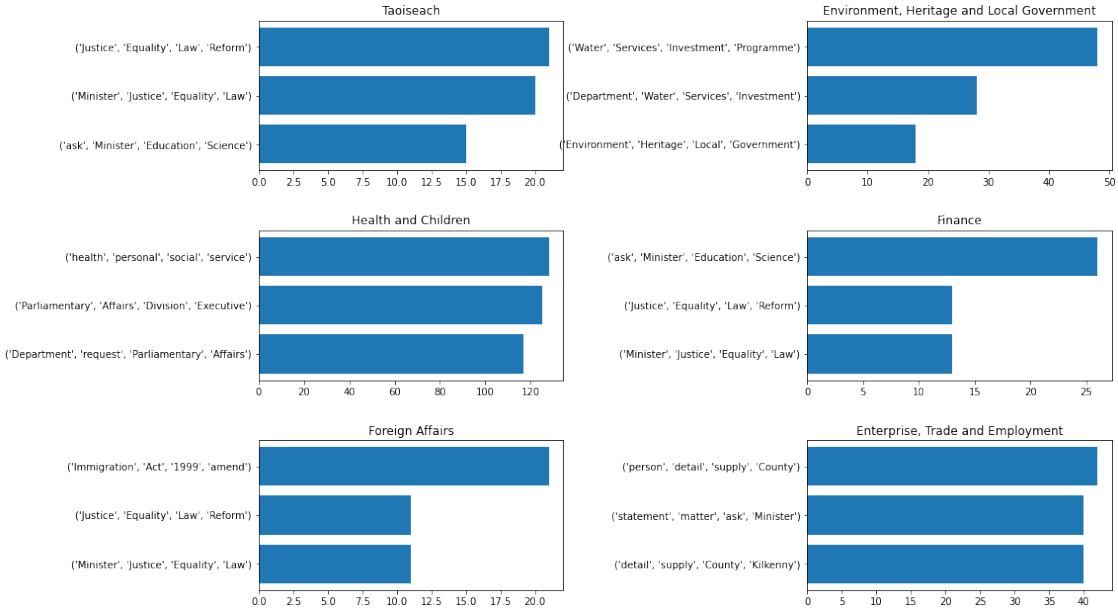


Widzimy, że najczęściej występujące słowa różnią się pomiędzy poszczególnymi departamentami. Przykładowo w ministerstwie finansów drugie najpopularniejsze słowo to podatek. Co ciekawe, znak euro pojawia się rzadziej w departamencie finansów niż w departamencie ochrony środowiska.

2.3.1 Aby znaleźć kontekst najbardziej popularnych słów, wykonujemy analizę ngramów - w naszym wypadku o długości 4.

[29]:
```
fig, ax = plt.subplots(3, 2)
fig.set_size_inches(15, 10)
a = 0

for i in range(3):
  for j in range(2):
    n_grams = ngrams(docs_by_deps[a], 4)
    n_grams_counter = Counter(n_grams).most_common(3)
    counts = pd.DataFrame(n_grams_counter, columns=['word', 'count']).
↪sort_values('count')
    counts.word = counts.word.apply(str)
    ax[i, j].barh(counts.iloc[:,0], counts.iloc[:, 1])
    ax[i, j].set_title(departments_names[a])
    a = a+1
```

```
plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.8,
                    hspace=0.4)
```



Możemy znaleźć informacje na temat konkretnych popularnych ustaw, np. Immigration Act 1999 lub programów - Water services investment programme

2.4 Analiza nazw własnych

[29]:
```
def label_entities(docs_speeches):
  ent_labels = [] #lista zawierająca etykiety nazw własnych
  all_ents = [] #lista nazw własnych
  for doc in docs_speeches:
    entities = doc.ents
    ent_label = [ent.label_ for ent in entities]
    ent_labels.append(ent_label)
    all_ents.append(entities)
  return all_ents, ent_labels
```

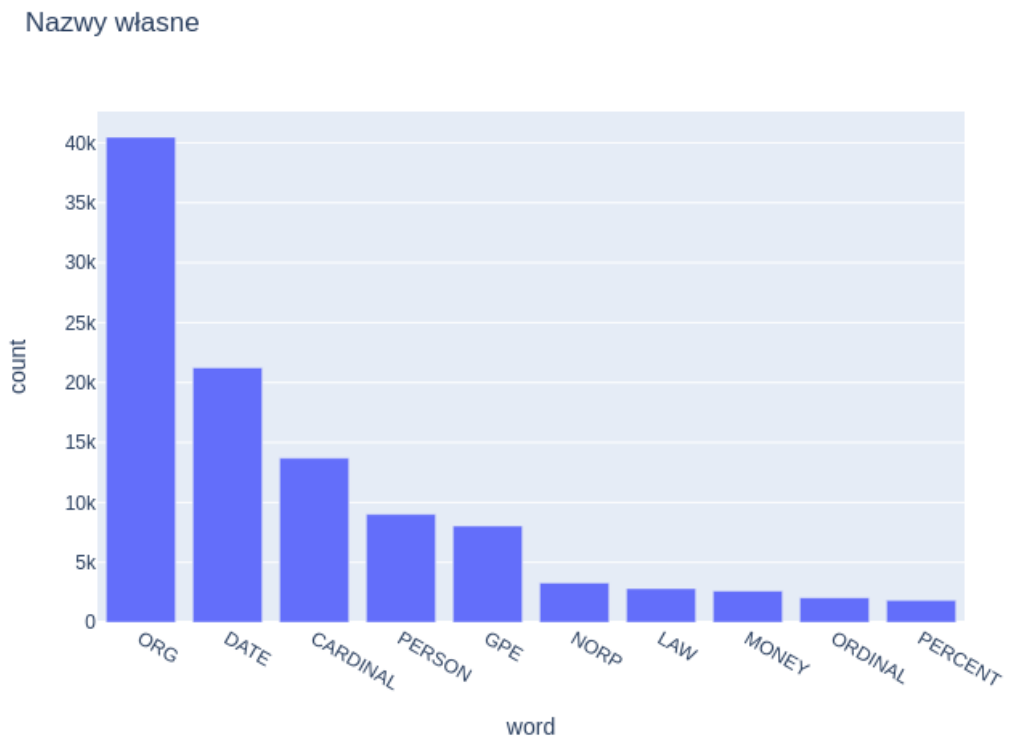
[30]:
```
[all_ents, ent_labels] = label_entities(docs_speeches)
```

[31]:
```
ents_by_deps = divide_by_departments(ent_labels, departments_names)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5:
VisibleDeprecationWarning:

Creating an ndarray from ragged nested sequences (which is a list-or-tuple of
lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If
you meant to do this, you must specify 'dtype=object' when creating the ndarray.
```

[32]:
```
counted_ents = Counter(list(itertools.chain(*ent_labels))).most_common(10)
counts = pd.DataFrame(counted_ents, columns=['word', 'count']).
 ↪sort_values('count', ascending = False)
fig = px.bar(counts, x = 'word', y = 'count', title = 'Nazwy własne')
img_bytes = fig.to_image(format="png")
from IPython.display import Image
Image(img_bytes)
```

[32]:



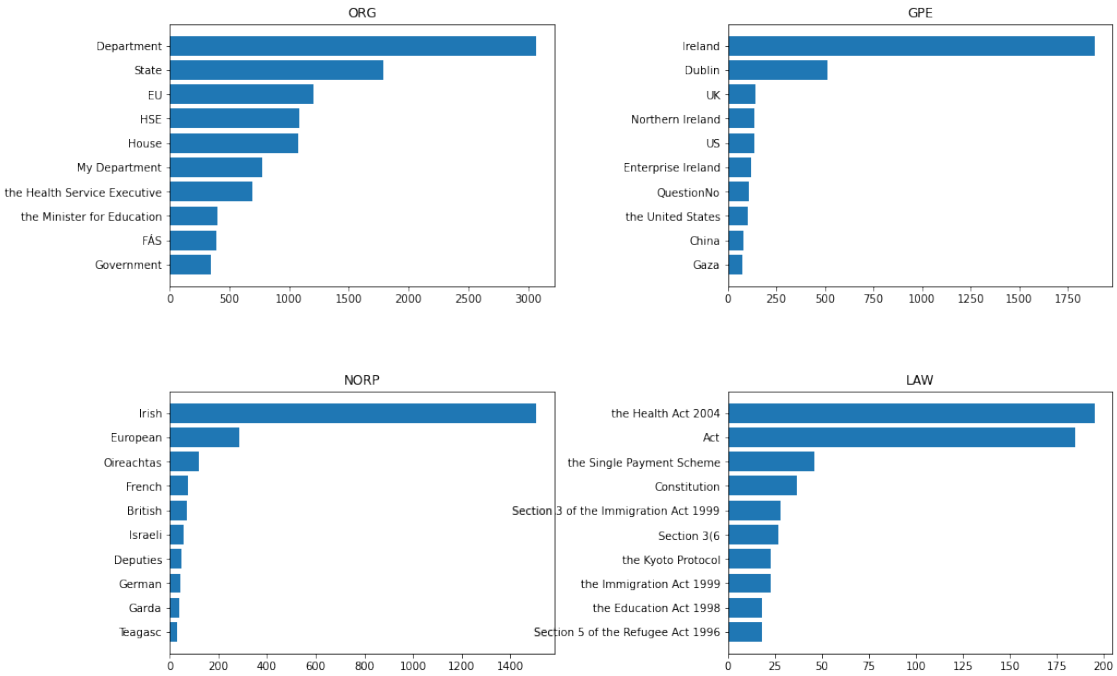
Najczęściej pojawiającymi się nazwami własnymi są nazwy organizacji i daty.

2.4.1 Zobaczmy, jakie organizacje, miejsca, narodowości oraz akty prawne są wymieniane najczęściej

[34]:
```
# zrobić dla wszystkich
ent_names = list(counts.word)
ent_names.reverse()
ent_names = ['ORG','GPE','NORP','LAW']
```

[35]:
```
fig, ax = plt.subplots(2, 2)
fig.set_size_inches(15, 10)
a = 0
n_grams_list = []

list_ent_labels = list(itertools.chain(*ent_labels))
for i in range(2):
  for j in range(2):
    is_org = list(map(lambda x: x == ent_names[a], list_ent_labels))
    org_list = list(np.array(list(itertools.chain(*all_ents)))[is_org])
    org_list = list(map(lambda x: str(x), org_list))
    counted_orgs = Counter(org_list).most_common(10)
    counts = pd.DataFrame(counted_orgs, columns=['word', 'count']).
 ↪sort_values('count')
    ax[i, j].barh(counts.iloc[:,0], counts.iloc[:, 1])
    ax[i, j].set_title(ent_names[a])
    a = a+ 1

plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.45,
                    hspace=0.4)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10:
VisibleDeprecationWarning:

Creating an ndarray from ragged nested sequences (which is a list-or-tuple of
lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If
you meant to do this, you must specify 'dtype=object' when creating the ndarray.
```



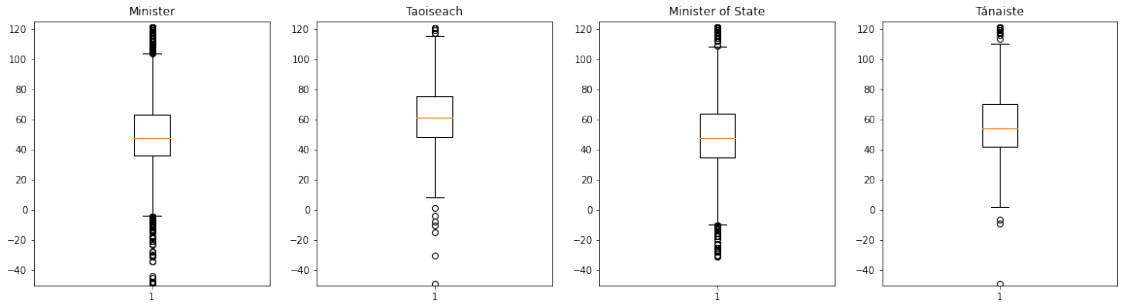
2.5 Sprawdzenie, czy pozycja mówcy ma wpływ na poziom skomplikowania wygłaszanego tekstu

W tym celu wykorzystamy Flesh reading ease score

80-90 easy to read
70-80 fairly easy to read
60-70 easily understood by 13- to 15-year-old students
50-60 fairly difficult to read

[36]:
```
merged_df['FRE'] = merged_df['speech'].apply(lambda x : flesch_reading_ease(x))
#top_6_departments['ARI'] = top_6_departments['speech'].apply(lambda x :␣
 ↪automated_readability_index(x))
positions = pd.unique(merged_df.position.dropna())
l = len(positions)
fig, ax = plt.subplots(1, l, figsize=(20, 5))

a = 0
for i in range(l):
    loc_dep = merged_df.loc[merged_df.position == positions[a]]
    ax[i].boxplot(loc_dep.FRE)
    ax[i].set_ylim([-50, 125])
    ax[i].set_title(positions[i])
    a = a+1
```

```
plt.show()
```



2.6 Chumra słów dla tytułów tekstów

[37]:
```
def tokenize_lemmatize(df_speech, docs_present = False):
  speeches_docs = []
  speeches = [] #tokenizacja i lematyzacja
  for speech in df_speech:
    doc = en(speech)
    tk = [token.lemma_ for token in doc if not token.is_stop if not token.
↪is_punct]
    speeches.append(tk)
    if docs_present:
      speeches_docs.append(doc)
  if docs_present:
    return speeches, speeches_docs
  return speeches
```

[38]:
```
bag_titles = tokenize_lemmatize(df2.title)
```

[39]:
```
counted_titles = Counter(list(itertools.chain(*bag_titles)))
```

[40]:
```
wc = WordCloud(width=800, height=400, colormap = 'Dark2')
wc.generate_from_frequencies(frequencies=counted_titles)
fig = plt.figure(1, figsize=(20, 12))
plt.imshow(wc)
plt.axis("off")
```

[40]:
```
(-0.5, 799.5, 399.5, -0.5)
```

Częstym motywem w tytułach wypowiedzi są pytania i odpowiedzi.

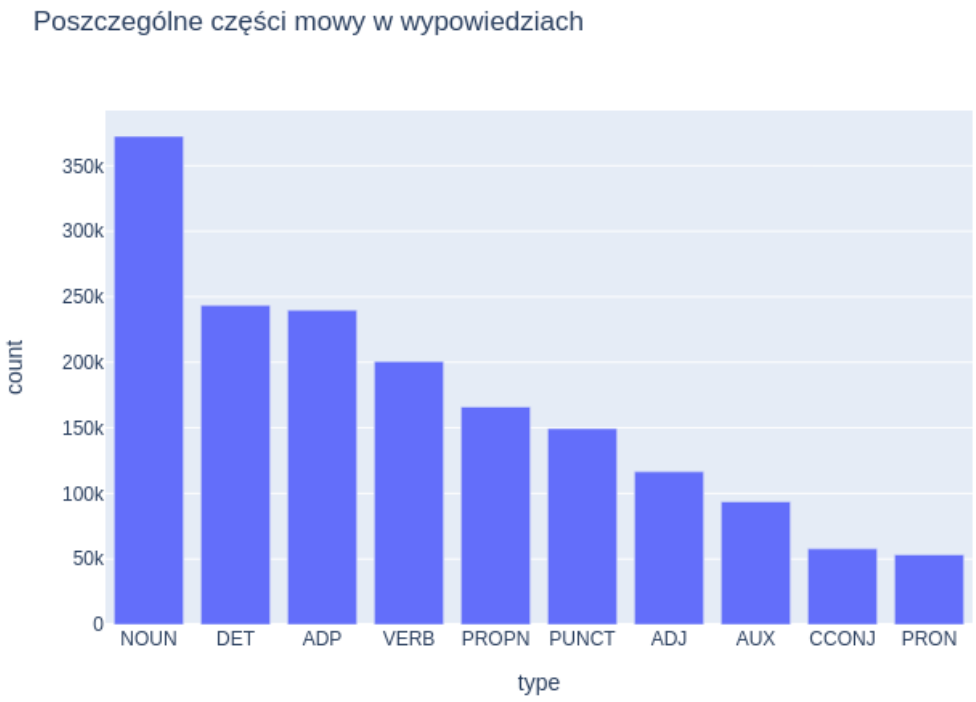
2.7 Analiza części mowy

[33]:
```
def get_pos_list(docs_speeches):
  positions_list = [] #lista zawierająca etykiety nazw własnych
  for doc in docs_speeches:
    for token in doc:
      pos = token.pos_
      positions_list.append(pos)
  return positions_list
```

[34]:
```
positions_list = get_pos_list(docs_speeches)
```

[35]:
```
pos_counter = Counter(positions_list).most_common(10)
pos_counter = pd.DataFrame(pos_counter, columns = ['type', 'count'])
fig = px.bar(pos_counter, x = 'type', y = 'count', title = 'Poszczególne części␣
 ↪mowy w wypowiedziach')
img_bytes = fig.to_image(format="png")
from IPython.display import Image
Image(img_bytes)
```

[35]:



Algorytm do wykrywania zależności

[44]:
```
sentences = list(docs_speeches[7].sents)
len(sentences)
```

[44]: 2

[45]:
```
mod_sentences = [] #zdania z czasownikami modalnymi
for sentence in sentences:
    for token in sentence:
        if token.dep_ == "aux" and token.tag_ == "MD":
            mod_sentences.append(sentence)
            break
# lemmas to lowercase
# for sentence in mod_sentences:
#     for token in sentence:
#         token.lemma_ = token.lemma_.lower()

mod_sentences
```

[45]: [The Minister of State should make no mistake about it.]

[46]:
```
def get_children_array(verb, sentence):
    children = []
    for token in sentence:
        if token.text == verb:
            for child in token.children:
                children.append(child)
    return unique(children)

# print(get_children_array('issue', sentences[0]))

def get_relation_between(word1, word2, sentence):
#     word1 = en(word1)[0].lemma_
#     print(word1)
    for token in sentence:
        if token.text == word1 and token.head.text == word2:
            relation = token.dep_
            print(token.text, token.head, token.dep_)
            return relation
    return None

# get_relation_between('The', 'issue', sentences[0])
def _get_all_conjucted_tokens(token):
    tokens = []
    for child in token.children:
        if child.dep_ == "conj":
            tokens.append(child)
    return tokens

def _get_coref_text(tokens):
    if len(tokens) == 0:
        return ""
    token = tokens[0]
    corefs = token._.corefs
    if len(corefs) == 0 or token.pos_ != "PRON":
        return ""
    return corefs[0]

def resolveCoreference(sub):
    """Znajduje podmiot, którego dotyczy zaimek (sub)."""
    return sub #duże problemy z pakietem neuralcoref:()
```

[47]:
```
def unique(list1):
    unique_list = []
    for x in list1:
        if x not in unique_list:
            unique_list.append(x)
    return unique_list
```

[48]:
```
def show_results(results_arr):
    attributes, objects, verbs = results_arr
    print('attributes: ', attributes)
    print('objects: ', objects)
    print('aims: ', verbs)
    print()
```

[49]:
```
def ig_tag(doc):
    """doc - the one sentence"""

    attributes, object, verbs = [], [], [] #'object' chyba nie powinno być w ogóle
    objects=[] #tego nie ma w pseudokodzie
    a = [token for token in doc if token.pos_ == "VERB"]
    verb = a[0] #nie wiem czy o to chodzi
    while verb is not None:
        attr = verb
        verb = None
        verbs.append(attr)

        #find subjects:
        newSubj = [child for child in attr.children if child.dep_ == "nsubj"]
        newPassiveSubj = [child for child in attr.children if child.dep_ == "nsubjpass"]

        if len(newSubj) == 0 and len(newPassiveSubj) == 0:
            attributes = [clausal for child in attr.children for clausal in child.children if child.dep_ == 'csubj']

        attributes = np.concatenate((attributes, newSubj), axis=None)
        objects = np.concatenate((objects, newPassiveSubj, [child for child in attr.children if child.dep_ == "dobj"]), axis=None)
        if attr.head.dep_ == 'conj' and attr.pos_ == 'VERB':
            verb = attr.head

    for subject in attributes:
        for attr in attributes:
            if attr.head.dep_ == 'conj':
                attributes.append(subject)
            attributes = np.concatenate((attributes, [child for child in subject.children if child.dep_ == "conj"]))
            if subject.pos_ == 'PRONOUN':
                subject = resolveCoreference(subject)
    for object in objects:
        object = np.concatenate((objects, [child for child in object.children if child.dep_ == "conj"]), axis=None)
```

```
    return unique(attributes), objects, verbs #aims=verbs?
# show_results(ig_tag(mod_sentences[1]))
```

[50]:
```
for sentence in mod_sentences:
    print(sentence)
    show_results(ig_tag(sentence))
```

```
The Minister of State should make no mistake about it.
attributes:  []
objects:  []
aims:  [should]
```

[36]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('EDA.ipynb')
```

```
File ‘colab_pdf.py’ already there; not retrieving.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

[NbConvertApp] Converting notebook /content/drive/MyDrive/Colab
Notebooks/EDA.ipynb to pdf
[NbConvertApp] Support files will be in EDA_files/
[NbConvertApp] Making directory ./EDA_files
[NbConvertApp] Making directory ./EDA_files
[NbConvertApp] Making directory ./EDA_files
[NbConvertApp] Making directory ./EDA_files
[NbConvertApp] Making directory ./EDA_files
[NbConvertApp] Making directory ./EDA_files
[NbConvertApp] Making directory ./EDA_files
[NbConvertApp] Making directory ./EDA_files
[NbConvertApp] Making directory ./EDA_files
[NbConvertApp] Making directory ./EDA_files
[NbConvertApp] Making directory ./EDA_files
[NbConvertApp] Writing 90008 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', './notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', './notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 778182 bytes to /content/drive/My Drive/EDA.pdf
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

[36]: 'File ready to be Downloaded and Saved to Drive'

```
** Koniec **
```