In [ ]:

```
!pip install textacy==0.8.0
!pip install neuralcoref --no-binary neuralcoref
!pip install swifter
!pip install spacy==2.1.0

!wget https://www.dropbox.com/s/v8xi3wopz0865jp/30_term.spacy
!wget https://www.dropbox.com/s/6d0645x9fx1wbdi/term30.csv
```

In [ ]:

```
!python -m spacy download en_core_web_sm
```

In [3]:

```
import spacy
import swifter
import pandas as pd
import numpy as np
import neuralcoref
import textacy

import matplotlib.pyplot as plt
import matplotlib
import plotly.express as px
import networkx as nx
import seaborn as sns
import nltk.sentiment
import random

from datetime import datetime
from tqdm.auto import tqdm
from wordcloud import WordCloud
from collections import Counter

%matplotlib inline
pd.options.mode.chained_assignment = None  # default='warn'
sns.set_theme(style="whitegrid")
pd.options.plotting.backend = "plotly"
```

```
100%|██████████| 40155833/40155833 [00:02<00:00, 13476574.79B/s]
```

Read data

In [4]:

```
df = pd.read_csv('term30.csv')
```

In [5]:

```
df = df.assign(
    year = pd.DatetimeIndex(df['date']).year,
    month = pd.DatetimeIndex(df['date']).month
).groupby(['year', 'month']). \
  apply(lambda group: group.sample(frac=.3, random_state=42)). \
  reset_index(drop=True). \
  drop(columns=['year', 'month'])
```

In [7]:

```
en = spacy.load('en_core_web_sm')
coref = neuralcoref.NeuralCoref(en.vocab)
en.add_pipe(coref, name='neuralcoref')
```

In [8]:

```
df['speech'] = df['speech'].swifter.apply(en)
```

Wordcloud

In [12]:

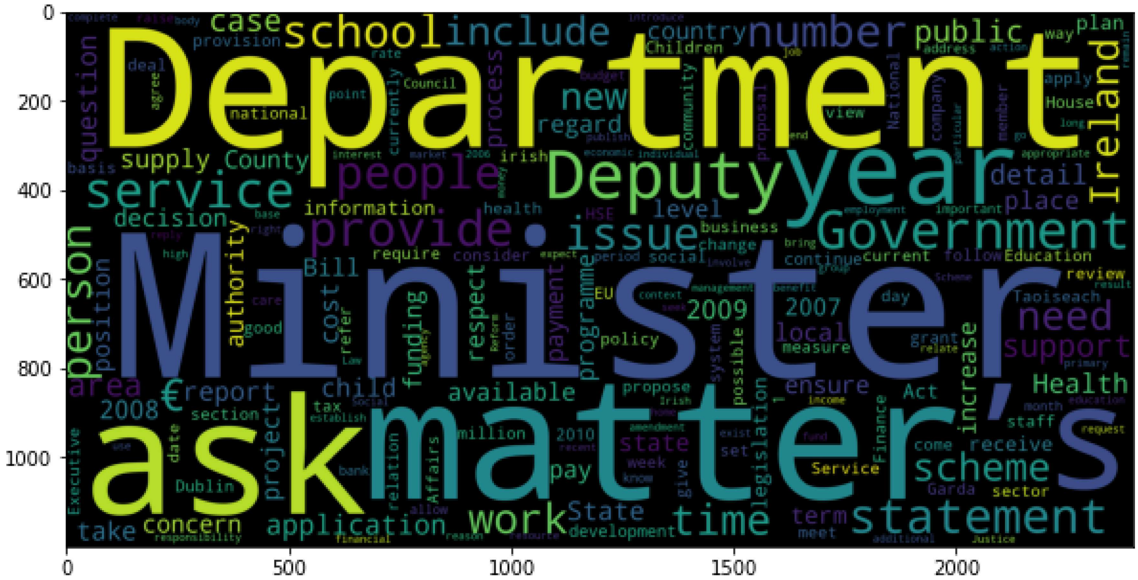
```
words = [token.lemma_ for doc in df['speech'] for token in doc
         if not token.is_stop and not token.is_punct]
word_counts = Counter(words)
```

In [13]:

```
wc = WordCloud(width=2400, height=1200)
wc.generate_from_frequencies(frequencies=word_counts)
plt.figure(figsize=(10, 8))
plt.imshow(wc)
```

Out[13]:

```
<matplotlib.image.AxesImage at 0x7f1bd51d76d0>
```



Najczęstsze słowa to te typowe dla polityki: `'Department'` , `'Minister'` itd. Chociaż pojawiają się słowa mniej oczywiste takie jak `'school'` , `'health'`

Wordcloud from keywords

In [14]:

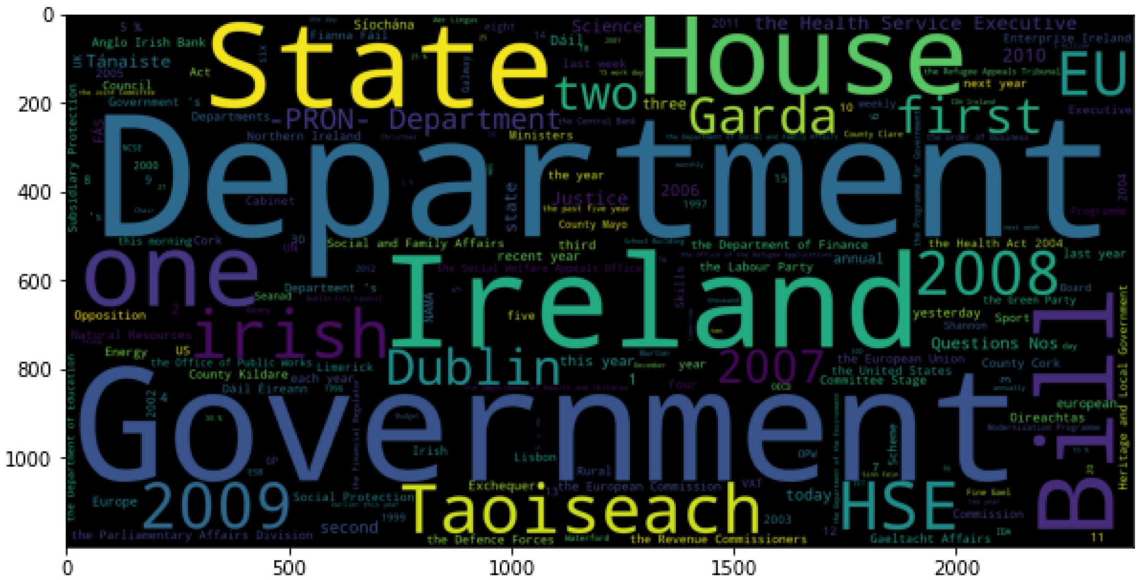
```
words = [token.lemma_ for doc in df['speech'] for token in doc.ents]
word_counts = Counter(words)
```

In [15]:

```
wc = WordCloud(width=2400, height=1200)
wc.generate_from_frequencies(frequencies=word_counts)
plt.figure(figsize=(10, 8))
plt.imshow(wc)
```

Out[15]:

```
<matplotlib.image.AxesImage at 0x7f1bd5202a10>
```



Most freqeuent bigram over time

In [16]:

```
df = df.assign(
    year = pd.DatetimeIndex(df['date']).year,
    quarter = (pd.DatetimeIndex(df['date']).month - 1)//3 + 1
)
```

In [20]:

```
def most_frequent_bigram(group):
  docs = group['speech']
  bigrams = [token.lemma_ for doc in docs for token in textacy.extract.ngrams(doc, 2, m
in_freq=2)]
  return Counter(bigrams).most_common(1)[0][0]
```

In [21]:

```
most_frequent_bigrams = df. \
  groupby(['year', 'quarter']). \
  apply(most_frequent_bigram). \
  reset_index(drop=False). \
  rename(columns={0: 'most_frequent_bigram'})
```

In [22]:

```
most_frequent_bigrams
```

Out[22]:

	year	quarter	most_frequent_bigram
0	2007	2	health care
1	2007	3	primary school
2	2007	4	Joint Committee
3	2008	1	local authority
4	2008	2	person concern
5	2008	3	person concern
6	2008	4	medical card
7	2009	1	person concern
8	2009	2	local authority
9	2009	3	person concern
10	2009	4	local authority
11	2010	1	local authority
12	2010	2	person concern
13	2010	3	person concern
14	2010	4	local authority
15	2011	1	local authority

Bigramy z przemów to przede wszyskim `'local authority'` i `'person concern'` . Jednakże widać pewne znaczące zdarzenia:

- Wypowiedź pewnego polityka o złym stanie szkół podstawowych w 2007
- Zamieszanie odnośnie kart medycznych w 2008

Mentioning of some important keywords by parties

In [23]:

```
important_keywords = ['health', 'education', 'defence', 'social', 'gun', 'capitalism',
'socialism', 'judge']
speeches_of_parties_count = df.groupby('party_name').size()
def get_most_frequent_party(keyword):
  speeches_with_keywords = df.loc[
      df['speech'].apply(lambda doc: keyword in [token.lemma_ for token in doc])
  ]
  speeches_counts = \
    speeches_with_keywords.groupby('party_name').size()/speeches_of_parties_count
  return speeches_counts.idxmax()
```

In [24]:

```
parties = list(map(get_most_frequent_party, important_keywords))
```

In [25]:

```
pd.DataFrame({'Keyword': important_keywords, 'Party': parties})
```

Out[25]:

	Keyword	Party
0	health	Fianna Fáil
1	education	Progressive Democrats
2	defence	Fianna Fáil
3	social	Progressive Democrats
4	gun	The Workers' Party
5	capitalism	The Workers' Party
6	socialism	Sinn Féin
7	judge	The Workers' Party

Which regions mention each other

In [28]:

```
regions = np.unique(df['const_name'])
regions
```

Out[28]:

```
array(['Carlow-Kilkenny', 'Cavan-Monaghan', 'Clare', 'Cork East',
       'Cork North-Central', 'Cork North-West', 'Cork South-Central',
       'Cork South-West', 'Donegal North-East', 'Donegal South-West',
       'Dublin (Clontarf)', 'Dublin (Finglas)', 'Dublin Central',
       'Dublin Mid West', 'Dublin North', 'Dublin North Central',
       'Dublin North-East', 'Dublin North-West', 'Dublin South',
       'Dublin South-Central', 'Dublin South-East', 'Dublin South-West',
       'Dublin West', 'Dún Laoghaire', 'Galway East', 'Galway North-East',
       'Galway West', 'Kerry North', 'Kerry South', 'Kildare',
       'Kildare North', 'Kildare South', 'Laoighis-Offaly',
       'Limerick East', 'Limerick West', 'Longford -Westmeath',
       'Longford-Roscommon', 'Louth', 'Mayo', 'Mayo West', 'Meath',
       'Meath East', 'Roscommon-South Leitrim', 'Sligo-Leitrim',
       'Sligo-North Leitrim', 'Tipperary North', 'Tipperary South',
       'Waterford', 'Westmeath', 'Wexford', 'Wicklow'], dtype=object)
```

In [29]:

```
regions_ids = dict(zip(regions, range(len(regions))))
```

In [30]:

```
def mentions_of_region(region_name):
  speeches_with_mentions = df.loc[
    df['speech'].apply(lambda doc: region_name in [token.lemma_ for token in doc.ents])
  ]
  return speeches_with_mentions.groupby('const_name').size()
```

In [32]:

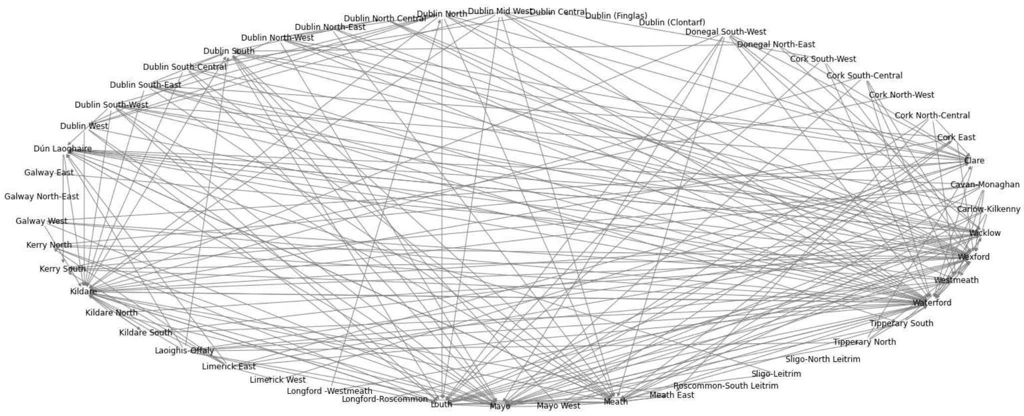
```
mentions_of_regions = list(zip(parties, map(mentions_of_region, parties)))
```

In [33]:

```
adj_matrix = np.zeros((len(parties), len(parties)))
for party, mentions in mentions_of_regions:
    for mention in mentions.items():
        adj_matrix[regions_ids[mention[0]], regions_ids[party]] = mention[1]
for i in range(adj_matrix.shape[0]):
  adj_matrix[i][i] = 0
```

In [35]:

```
graph = nx.from_numpy_matrix(adj_matrix, create_using=nx.DiGraph)
plt.figure(figsize=(24, 10))
nx.draw_circular(graph,
                 labels=dict(zip(regions_ids.values(),regions_ids.keys())),
                 node_color='#00000000',
                 edge_color='gray')
```



Najczęściej wspominane okręgi wyborcze:

- Mayo and related
- Louth
- Dun Laoghaire
- Kildare
- ...

Create tables with external data

Dzielimy przemawiających na 3 grupy:

- other/ordinary - ci którzy nie zajmują w momencie przemawiania żadnego stanowiska, jest ich 134
- minsters - ministrowie, ministrowie stanu (minister of state), wicepremierzy, jest ich 37
- taoiseach - premierzy. W rozpatrywanej kadencji było dwóch

In [ ]:

```
docs = df['speech']
```

In [ ]:

```
df_ministers=pd.read_table("Dail_debates_1937-2011_ministers.tab")
```

In [ ]:

```
df_big=pd.merge(df,df_ministers,on='memberID',how='left')
df_mins=df_big.loc[~(df_big["start_day"].isna())]
df_mins["end_date"] = df_mins["end_date"].fillna("2012-01-01")
```

In [ ]:

```
df_mins["end_date"]= pd.to_datetime(df_mins["end_date"], format="%Y-%m-%d")
df_mins["start_date"]= pd.to_datetime(df_mins["start_date"], format="%Y-%m-%d")
df_mins["date"]= pd.to_datetime(df_mins["date"], format="%Y-%m-%d")
```

In [ ]:

```
df_mins=df_mins.loc[df_mins["date"]<df_mins["end_date"]]
df_mins=df_mins.loc[df_mins["start_date"]<df_mins["date"]]
```

In [ ]:

```
df_other=df[~df["speechID"].isin(df_mins["speechID"])]
```

In [ ]:

```
df_other_sm=df_other.sample(frac=0.25)
```

In [ ]:

```
df_taos=df_mins.loc[df_mins["position"]=='Taoiseach']
```

In [ ]:

```
df_mins=df_mins[df_mins.position!="Taoiseach"]
```

In [ ]:

```
df_mins=df_mins.drop_duplicates(["speechID"])
```

In [ ]:

```
df['copy_index'] = df.index
df_mins=df.merge(df_mins, on="speechID", how="right")
```

In [ ]:

```
df_taos=df.merge(df_taos, on="speechID", how="right")
```

Create subdocs

Dla każdej grupy tworzymy osobne liste docs ze spacy

In [ ]:

```
docs_other=[]
for i in df_other.index.tolist():
    docs_other.append(docs[i])
```

In [ ]:

```
docs_mins=[]
for i in df_mins.copy_index:
    docs_mins.append(docs[i])
```

In [ ]:

```
docs_taos=[]
for i in df_taos.copy_index:
    docs_taos.append(docs[i])
```

Average length of a speech

In [ ]:

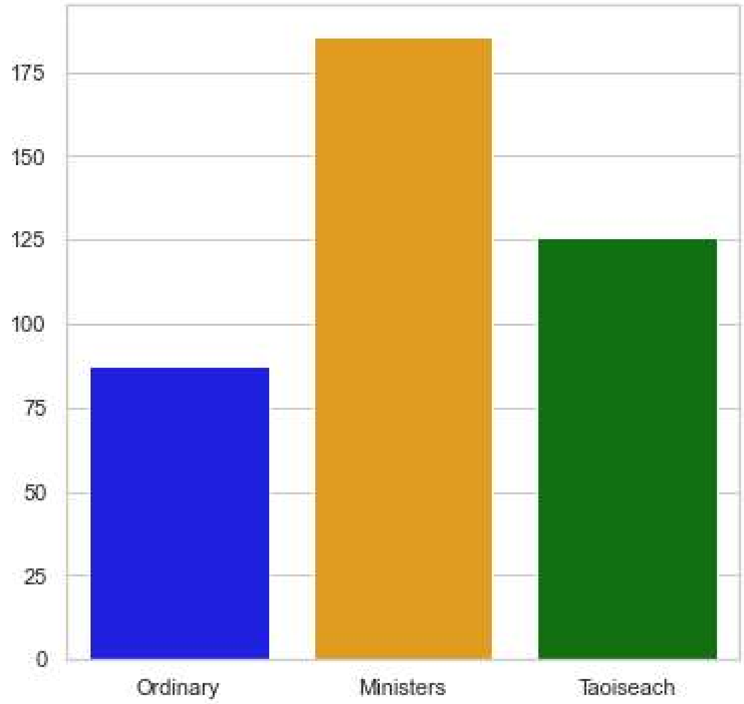
```
lens_other=[len(i) for i in docs_other]
lens_mins=[len(i) for i in docs_mins]
lens_taos=[len(i) for i in docs_taos]
```

In [ ]:

```
length_other = sum(lens_other)/len(lens_other)
length_mins = sum(lens_mins)/len(lens_mins)
length_taos = sum(lens_taos)/len(lens_taos)
```

In [ ]:

```
plt.figure(figsize=(6,6))
ax = sns.barplot(y=[length_other, length_mins, length_taos], x=["Ordinary", "Ministers"
, "Taoiseach"],
                 palette=["blue", "orange", "green"], orient="v")
```

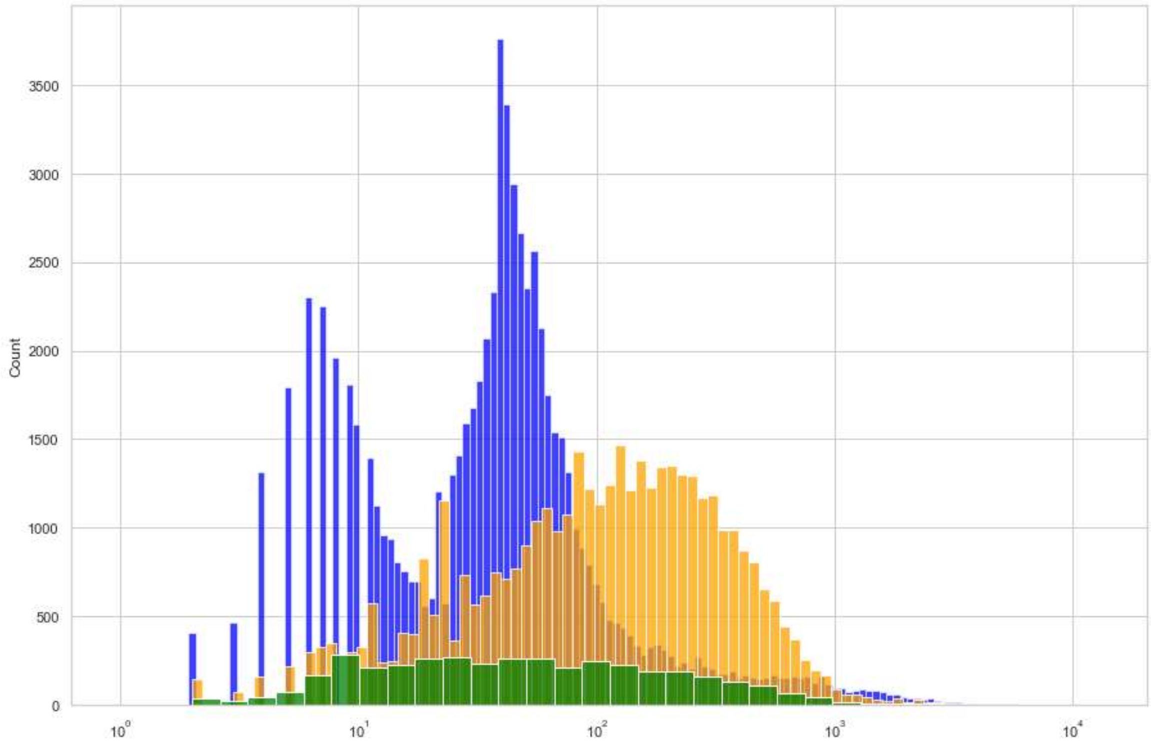


Co ciekawe, przemówienia premiera są znaczenie krótsze od przemówień mistrów.

In [ ]:

```
plt.figure(figsize=(15,10))
sns.histplot(lens_other, log_scale=True, color="blue")
sns.histplot(lens_mins, log_scale=True, color="orange")
sns.histplot(lens_taos, log_scale=True, color="green")
```

```
<AxesSubplot:ylabel='Count'>
```



Widać dwa peaki w rozkładzie czasu przemówień zwykłych deputowanych - może to kwestia trybu przemawiania, np zadawania pytań/komentarza/odpowiedzi?

Wordclouds

In [ ]:

```
lemmas_other=[]
for i in range(len(docs_other)):
    for t in [token.lemma_ for token in docs_other[i] if not token.is_stop if not token
.is_punct]:
        lemmas_other.append(t)
lemmas_mins=[]
for i in range(len(docs_mins)):
    for t in [token.lemma_ for token in docs_mins[i] if not token.is_stop if not token.
is_punct]:
        lemmas_mins.append(t)
lemmas_taos=[]
for i in range(len(docs_taos)):
    for t in [token.lemma_ for token in docs_taos[i] if not token.is_stop if not token.
is_punct]:
        lemmas_taos.append(t)
```

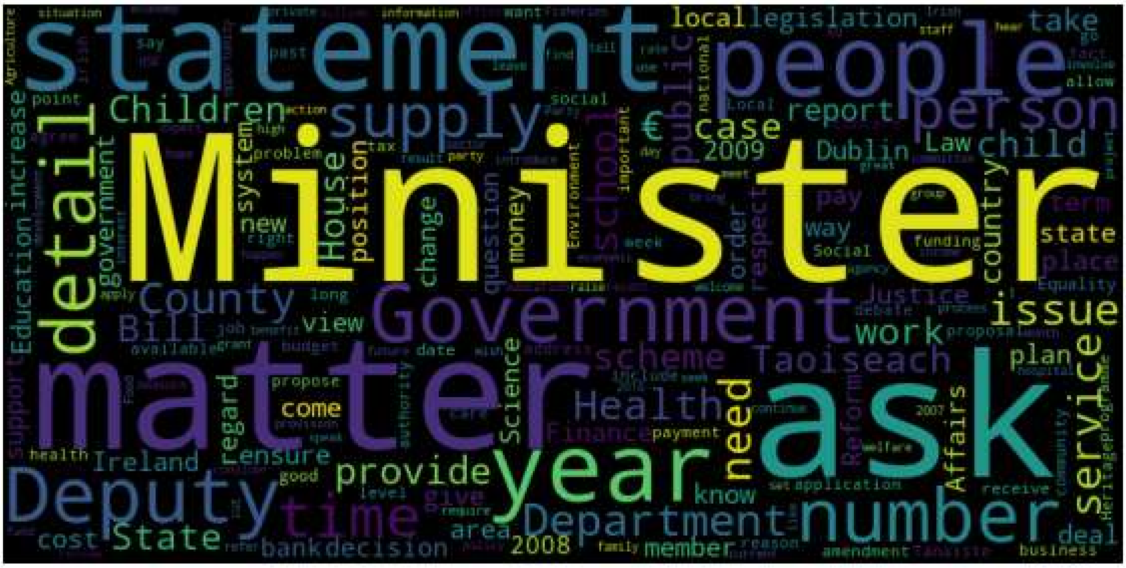
In [ ]:

```
from collections import Counter
word_counts_other = Counter(lemmas_other)
word_counts_mins = Counter(lemmas_mins)
word_counts_taos = Counter(lemmas_taos)
```

In [ ]:

```
wc = WordCloud(width=800, height=400)
wc.generate_from_frequencies(frequencies=word_counts_other)
plt.figure(figsize=(10,8))
plt.axis("off")
plt.imshow(wc)
```

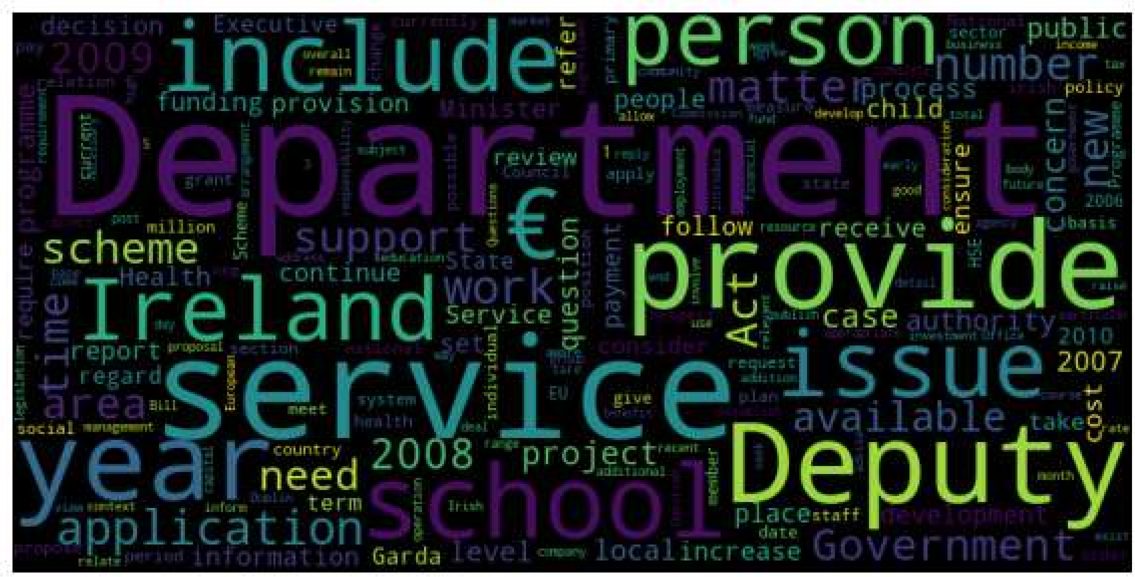
```
<matplotlib.image.AxesImage at 0x194378e9400>
```



In [ ]:

```python
wc = WordCloud(width=800, height=400)
wc.generate_from_frequencies(frequencies=word_counts_mins)
plt.figure(figsize=(10,8))
plt.axis("off")
plt.imshow(wc)
```

<matplotlib.image.AxesImage at 0x194371a17c0>



In [ ]:

```
wc = WordCloud(width=800, height=400)
wc.generate_from_frequencies(frequencies=word_counts_taos)
plt.figure(figsize=(10,8))
plt.axis("off")
plt.imshow(wc)
```

<matplotlib.image.AxesImage at 0x19436960ac0>

Widać, kto jest adresatem czyjej wypowiedzi. Premier najczęściej zwraca się do deputowanych, a zwykli deputowani - do ministrów. Ministrowie najczęściej mówią o ministerstwach - może podczas składania sprawozdań z pracy ich departamentu? Szeregowi posłowie też najczęściej zadają pytania w swoich wypowiedziach - niektóre krótkie pytania są zapisane w danych w mowie zależnej, stąd słowo "ask" częste wśród wypowiedzi.

basic Sentiment Analysis

In [ ]:

```
from nltk.sentiment import SentimentIntensityAnalyzer

def get_scores(docs_sample):

    score={"neg":0, "neu":0, "pos":0, "compound":0}
    for i in range(len(docs_sample)):
        sia = SentimentIntensityAnalyzer()
        ss=sia.polarity_scores(str(docs_sample[i]))
        for k in score.keys():
            score[k]+=ss[k]
    for k in score.keys():
            score[k]=score[k]/len(docs_sample)
    return score
```

In [ ]:

```
docs_other_sample=random.sample(docs_other, 1000)
docs_mins_sample=random.sample(docs_mins, 1000)
docs_taos_sample=random.sample(docs_taos, 1000)
score_other=get_scores(docs_other_sample)
score_mins=get_scores(docs_mins_sample)
score_taos=get_scores(docs_taos_sample)
```

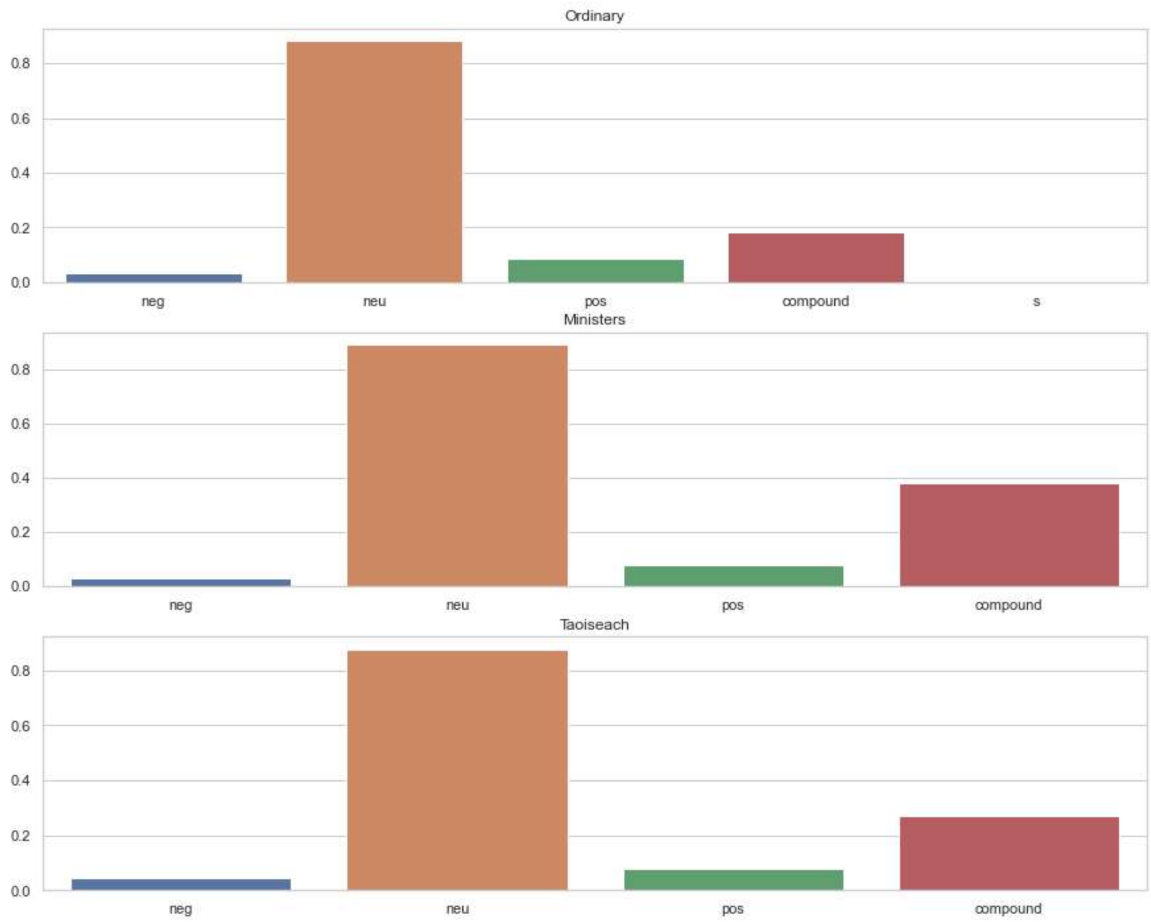
In [ ]:

```
dics=[score_other, score_mins, score_taos]
tempo={"group":0, "neg":0,  "pos":0, "compound":0}
for i in dics:
    for k in i.keys():
        tempo[k]=i[k]
```

In [ ]:

```
fig, ax = plt.subplots(3, figsize=(15,12))
sns.barplot(x=list(score_other.keys()), y=list(score_other.values()), ax=ax[0]).set_tit
le("Ordinary")
sns.barplot(x=list(score_mins.keys()), y=list(score_mins.values()), ax=ax[1]).set_title
("Ministers")
sns.barplot(x=list(score_taos.keys()), y=list(score_taos.values()), ax=ax[2]).set_title
("Taoiseach")
```

```
Text(0.5, 1.0, 'Taoiseach')
```



Generalnie wszystkie wypowiedzi są klasyfikowane raczej jako neutralne. Wielkość "compound" jest zdecydowanie najniższa u zwykłych posłów - jest to wielkość mówiaca o ogólnym wydźwięku wypowiedzi, nie do końca bezpośrednio zależna od pozostałych wielkości. Jest ona jednak dalej dodatnia - compund przyjmuje wartości na przedziale (-1, 1).

Name mentions

In [ ]:

```
df_mins = df_mins.loc[:,~df_mins.columns.duplicated()]
df_taos = df_taos.loc[:,~df_taos.columns.duplicated()]
```

In [ ]:

```
names_other=list(set([i.split()[-1] for i in df_other.member_name]))[2:]
names_mins=list(set([i.split()[-1] for i in df_mins.member_name_x]))[2:]
names_taos=list(set([i.split()[-1] for i in df_taos.member_name_x]))
```

In [ ]:

```
sample_texts=random.sample(docs, 10000)
```

In [ ]:

```
dic_names_other={}
for i in names_other:
    dic_names_other[i]=0
dic_names_mins={}
for i in names_mins:
    dic_names_mins[i]=0
dic_names_taos={}
for i in names_taos:
    dic_names_taos[i]=0
```

In [ ]:

```
for i in sample_texts:
    words=str(i).split()
    for w in words:
        if w in dic_names_taos.keys():
            dic_names_taos[w]+=1
        if w in dic_names_mins.keys():
            dic_names_mins[w]+=1
        if w in dic_names_other.keys():
            dic_names_other[w]+=1
dic_names_mins.pop("White")
dic_names_other.pop("White")
```

```
90
```

Usuwamy panią White z naszych nazwisk - może ona co prawda być często wspominana, bo w 2010 objęła stanowisko w ministerstwie stanu, ale jej nazwisko może też pojawiać się w innch konktekstach.

Average number of name mentions per group

In [ ]:

```
print("Ordinary: ", sum(dic_names_other.values())/len(dic_names_other.keys()))
print("Ministers: ", sum(dic_names_mins.values())/len(dic_names_mins.keys()))
print("Taoiseach: ", sum(dic_names_taos.values())/len(dic_names_taos.keys()))
```

```
Ordinary:  12.955223880597014
Ministers:  8.432432432432432
Taoiseach:  22.0
```

Most mentioned

In [ ]:

```
t=max(dic_names_taos, key=dic_names_taos.get)
m=max(dic_names_mins, key=dic_names_mins.get)
o=max(dic_names_other, key=dic_names_other.get)
```

Most menionen non minister:

Joan Burton. Przewodnicząca Partii Pracy. Ciekawe o tyle, że w następnej kadencji już została ministrem.

Most mentioned minister:

Micheal Martin, minister zatrudnienia a potem mister spraw zewnętrznych.

Most mentioned Taoiseach:

Co ciekawe, Brian Cohen był wspominany dużo rzadziej niż Bertie Ahern, mimo że to on przez większość kadencji zajmował stanowisko premiera.

IG tagging

In [ ]:

```
# load NeuralCoref and add it to the pipe of SpaCy's model
import neuralcoref
coref = neuralcoref.NeuralCoref(en.vocab)
en.add_pipe(coref, name='neuralcoref')
```

In [ ]:

```
def find_deontic(sent):
  deontic_verbs = ['can', 'may', 'must', 'shall', 'could', 'might', 'should
  deontic = None
  for token in sent:
    if token.lemma_ in deontic_verbs:
      deontic = token
  return deontic

def get_children_with_dep(token, dep:str):
  return [c for c in token.children if c.dep_ == dep]

def get_clausual_subject(token):
  csubjs = get_children_with_dep(token, 'csubj')
  out = list()
  for c in csubjs:
    out = out + get_children_with_dep(c, 'nsubj')
  return out

def get_coref(token):
  corefs = token._.corefs
  if len(corefs) == 0 or token.pos_ != "PRON":
      return None
  return corefs[0]


def tag_ig(doc):
  sents = list(doc.sents)
  ig_deontics = list()
  ig_attributes = list()
  ig_objects = list()
  ig_aims = list()
  for sent in sents:
    d = find_deontic(sent)
    if d is None:
      return pd.Series([None ,None, None, None], index=['ig_deontic','ig_at
    attributes = list()
    objects = list()
    verbs = list()
    verb = d.head
    while verb is not None:
      attr = verb
      verb = None
      verbs.append(attr)
      newSubj = get_children_with_dep(attr, 'nsubj')
      newPassiveSubj = get_children_with_dep(attr, 'nsubjpass')
      if len(newSubj) == 0 and len(newPassiveSubj)==0:
        attributes = get_clausual_subject(attr)
      attributes = attributes + newSubj
      objects = objects + newPassiveSubj + get_children_with_dep(attr, 'dok

      if attr.dep_ == 'conj' and attr.pos_ == 'VERB':
        verb = attr.head
    for subject in attributes:
      if subject.dep_ =='conj':
        attributes.append(subject)
      attributes = attributes + get_children_with_dep(subject, 'conj')
      if subject.pos_ == "PRON":
        subject = get_coref(subject)
    for object_ in objects:
      objects = objects + get_children_with_dep(object_, 'conj')

    ig_deontics.extend([d.lemma_])
    ig_attributes.extend(attributes)
    ig_objects.extend(objects)
    ig_aims.extend(verbs)
```

```
    return pd.Series([ig_deontics, ig_attributes, ig_objects, ig_aims], index
```

Pobieranie tekstów zawierających modal verbs.

In [ ]:

```
!wget -O deontic_df.csv https://f003.backblazeb2.com/b2api/v1/b2_download_f
df = pd.read_csv('deontic_df.csv')
df = df.sample(n=3000, random_state=42)
df['speech_spacy'] = df['speech'].swifter.apply(en)
```

In [83]:

```
df[['ig_deontic', 'ig_attributes', 'ig_objects', 'ig_aims']]= df['speech_sp
df = df[~df.ig_deontic.isnull()]
df = df.explode('ig_deontic').explode('ig_attributes').explode('ig_objects'
```

In [ ]:

```
# dodanie atrybutu, czy mówca jest z partii rządzącej
df['ruling_party'] = df['partyID'].map(lambda x: 1 if x in [8,10,21] else (
```

In [130…

```
df[['ruling_party','ig_deontic', 'ig_attributes', 'ig_objects', 'ig_aims']]
```

Out[130…

	ruling_party	ig_deontic	ig_attributes	ig_objects	ig_aims
6755	0	can	They	loan	get
5253	0	must	Deputy	question	ask
1597	0	may	Minister	NaN	go
8248	0	can	NaN	that	applied
7711	0	may	We	House	leave

Wykres poniżej przedstawia liczbę zwrotów z *modal verbs* z podziałem na partie rządzące oraz opozycyjne. Największe dysproporcje widać przy czasowniku *should* (częściej wypowiadane przez partie rządzące) oraz *must* (częściej wypowiadane przez partie opozycyjne).

In [146…

```
import matplotlib.pyplot as plt
pd.options.plotting.backend = 'matplotlib'
df.groupby(['ig_deontic', 'ruling_party'])['ig_deontic'].count().to_frame()
plt.legend(['Partie rządzące', 'Partie opozycyjne'])
plt.show()
```