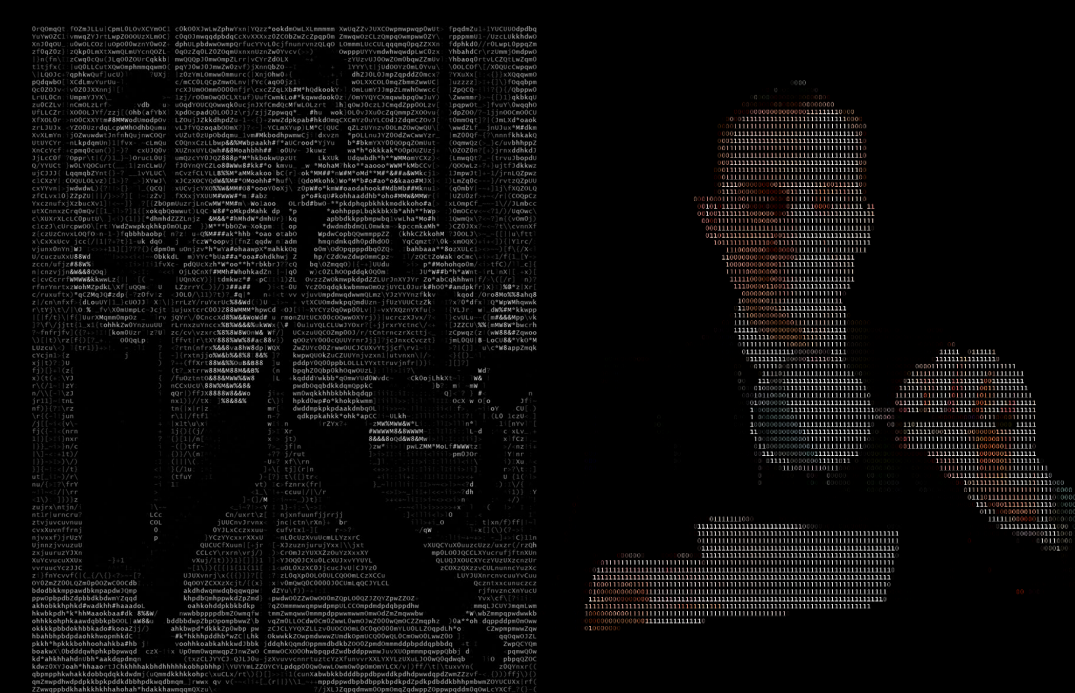
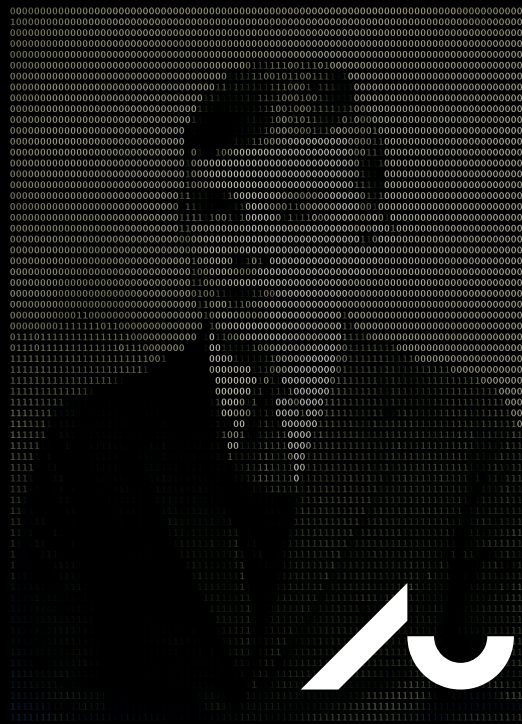
Tæl din tekst

Introduktion til digital tekstanalyse

Screen-shots med eksempler
på programmeringsopgaver
lavet i Jupyter Notebooks

De interaktive opgaver fra kurset findes
her: https://github.com/ulfberthelsen





Tæl din tekst: Basic Python 1

Keywords: `strings`, `lists`, `variabler`, `assignment`, `concatenation`, `funktion`, `methods`

Nye Python-udtryk: `print()`, `len()`, `=`, `+`, `.upper()`, `.lower()`, `.strip()`, `.split()`, `.replace()`

Tekst som strings

Når vi skal arbejde med tekst i Python, møder vi næsten altid teksten som et `string object`, dvs som en instans af Pythons `string class` (dvs. skabelonen for alle string objects). Så længe teksten har datatypen `string` kan vi foretage forskellige operationener på teksten, ved at anvende forskellige `string methods`. Husk:

- Strings skrives altid mellem anførselstegn ('_' eller " _ ")
- Python opfatter en `string` som en ubrudt kæde af tegn.
- Python behandler blanktegn, punktummer, kommaer og andre tegn på samme måde som store og små bogstaver, når de optræder i en `string`.

Vi kan enten importere teksten ved at åbne en tekst-fil, vi har gemt på computeren (det kommer vi til senere), eller vi kan taste teksten direkte ind og gemme den midlertidigt under et variabelnavn (hardcoding data).

I det følgende skal I bruge disse to teksteksempler. Husk at få blank tegn og punktummer med - dem skal vi bruge i en af øvelserne :-)

Teksteksempel 1: "... Computational thinking dækker kort sagt over tænkemåder, der gør os i stand til at handle i verden gennem computere med henblik på at gøre ting vi ellers ikke ville have været i stand til at gøre."

Teksteksempel 2: "Begrebet computational thinking er ikke nyt, men det blev for alvor sparket ind i uddannelsesdebatten i 2006, da den amerikanske datalogi-professor Jeanette Wing (2006) formulerede et uddannelsespolitisk opråb med henblik på at få flere unge til at søge ind på universiteternes datalogiuddannelser ..."

Vigtigt: Det er ok at copy/paste teksteksempler, men det er vigtigt, at I selv indtaster kode-eksemplerne, så I får det op gennem fingrene!!

Opgave 2

Når vi skal analysere digitale tekster, er det ofte nødvendigt at rense og formattere teksterne på forskellige måder. Denne del af arbejdet omtales ofte som `data cleaning` og `pre-processing`. Datacleaning og preprocessing kan være komplekse og tidskrævende processer, hvis der er tale om store tekst-korpusser. Kompleksiteten er imidlertid ofte et udtryk for, at man gør mange forskellige ting ved sine tekster, inden de er klar til analyse. De enkelte skridt i processen er ofte simple.

I denne opgave skal I afprøve nogle simple `string methods` (dvs. funktioner der kun kan anvendes på `string objects`), der kan være nyttige, når tekster skal renses og klargøres.

Husk at **methods** noteres med punktum-notation, fx `variabelnavn.method_x()` (læses anvend `method_x` på indholdet af `variabelnavn`).

Prøv at anvende følgende methods på den sammenføjede tekst. Husk at gemme outputtet i en ny variabel. Print reultatet, så I kan følge med i, hvad der sker.

Eksempel:

```
variabelnavn_4 = variabelnavn_3.upper()
print(variabelnavn_4)
```

Nyttige string methods:

1. `.upper()` : konverterer alle bogstaver til store bogstaver
2. `.lower()` : knoverterer alle bogstaver til små bogstaver
3. `.strip()` : fjerner tegn i begyndelsen og slutningen af strengen - tegnet angives i anførselstegn i parentesen, fx `.strip('.')`
4. `.rstrip()` : fjerner kun tegn til højre i strengen (r for right)
5. `.lstrip()` : fjerner kun tegn til venstre i strengen (l for left)
6. `.replace(arg1, arg2)` : erstatter `argument 1` men `argument 2` fx, `.replace(".", " ")`

I kan bruge koden `dir("")` til at se en samlet oversigt over `string methods`.

Hvis I bruger koden `print(variabelnavn)` printes ren tekst. Hvis I blot taster `variabelnavn` printes indholdet af variablen. Afprøv begge dele, og overvej forskellen.

In [ ]:

Tæl din tekst: Basic Python 3

Definér en funktion

Keywords: `definition`, `funktion`

Nye Python-udtryk: `def`, `return`

At **definere** en **funktion** vil sige at pakke en eller flere kodesekvenser sammen under ét navn, så det kan let kan anvendes, gemmes og genbruges. I kan tænke på det som en funktionel pedant til variabler. Vi kan gemme komplekse datastrukterer under et simpelt vaiabelnavn. På samme måde kan vi gemme kompleks funktionalitet under et simpelt funktionsnavn. Fordelen ved funktioner er, dels at man let kan genbruge funktionalitet, dels at det er lettere at finde fejl, idet man ikke behøver at tjekke al kode, men blot koden i den funktion, der fremprovokerer fejlen.

Definitioner består af tre dele: 1) Definitionskommandoen, 2) en kodeblok, 3) et `return`-kommando.

Definitionskommandoen består kommandoen `def`, et `funktionsnavn` samt en eller flere pladsholdere for `argumenter`. Udtrykket `def` fortæller Python, at den efterfølgende kodeblok skal gemmes under det `funktionsnavn`, der specificeres. Funktionsnavnet er valgfrit. Når funktionen efterfølgende skal bruges, kalder (call into action) man funktionen med kommandoen `funktionsnavn()`. En funktion kan have en række `parametre`, dvs. tage inputs, hvis værdier påvirker outputtet. De konkrete værdier, der indsættes i funktionen kaldes `argumenter` og specificeres i parentesen umiddelbart efter funktionsnavnet. Husk at afslutte definitionskommandoen med `:`.

Syntaksen ser således ud: `def funktion(argument_1, argument_2, argument_x):`

Kodeblokken består af de kodelinjer, man ønsker at gemme under det angivene `funktionsnavn`. Kodeblokken indenteres (indrykkes med fire blanktegn) på linjen under definitionskommandoen.

Return-kommandoen angiver hvilket output, funktionen skal returnere. Hvis intet specificeres er return-typen **NONE**. Funktionen returnerer det, der specificeres efter udtrykket `return`.

Opgave 1

I eksemplet nedenfor defineres en lille simpel funktion. Funktionen får navnet `printStreng`. Når funktionen anvendes indtasten en værdi i parentesen. Fordi kodeblokken indeholder et `print()` -statement, skal inputtet vær af typen `string`. Inputtet gives automatisk det variabelnavn, der i definitionskommandoen specificeres i parentesen - i dette tilfælde navnet 'streng'. Kodeblokken indeholder et for-loop, der looper over elementerne i inputtet, dvs. bogstaverne i den `string`, som funktionen anvendes på. For hvert eklement i inputtet (for hvert bogstav i strengen) printes dette element.

Prøv at afvikle koden nedenfor.

In [1]:
```
def printStreng(streng):
    for b in streng:
        print(b)
```

In [3]:
```
txt = "Der var en gang"
printStreng(txt)
```

```
D
e
r

v
a
r

e
n

g
a
n
g
```

Tæl din tekst: Basic Python 4

Tekst som lister

Keywords: `pre-processing`, `funktioner`, `loop`, `lister`, `tekster`, `kapitler`, `sætninger`, `ord`, `indeks`

Nye Python-udtryk: `range`, `[0:4]`

Der er mange måder at håndtere tekster på, når man arbejder digitalt. I det følgende skal vi kigge på to måder at håndtere en længere tekst på. Eksemplet er Herman Bangs novellesamling *Exentriske Noveller*. I begge eksempler splittes teksten først i kapitler.

Forberedelse

Som forberedelse til arbejde skal vi første definere to `funktioner`. Den første af de to funktioner er identisk med `rens`-funktionen, vi brugte sidst. Her blot omdøbt til `rens_ord`. Den anden funktion er magen til, bortset fra at den splitter ved punktum i stedet for ved blanktegn. Denne funktion kalder vi `rens_sæt`.

Vi skal desuden have åbnet og indlæst filen. Her bruger vi funktionerne `open()` og `.read()` ligesom sidst.

Definition af `rens_ord`

In [1]:
```
def rens_ord(text_0):
    text_1 = text_0.replace("\n"," ")
    text_2 = text_1.replace("."," ")
    text_3 = text_2.replace(","," ")
    text_4 = text_3.replace(":"," ")
    text_5 = text_4.replace("*"," ")
    text_6 = text_5.replace("-"," ")
    text_7 = text_6.replace("!"," ")
    text_8 = text_7.replace("“"," ")
    text_ren = text_8.replace("-"," ")
    text_lav = text_ren.lower()
    text_token = text_lav.split()
    return text_token
```

Definition af `rens_sæt`

Pandas 2: Workflow og loops over dataframes

Keywords: `work flow`, `data frames`, `loops`, `gem som csv`

Nye Python-udtryk: `.loc[]`, `.reindex`, `.apply()`, `.describe()`, `.to_csv()`, `.iterrows()`, `.iteritems()`, `.map()`

Når vi skal arbejde med digital tekstanalyse, er der mange fordele ved at arbejde med teksterne i `data frames`. Dels giver det en overskuelige håndtering tekster og metadata, dels giver det stordriftsfordele i den forstand, at det er let at gennemføre operationer på mange tekster på en gang og efterfølgende gemme reultaterne i nye kolonner.

Nedenfor gennemgår vi et minieksempel på, hvordan et komplet `work flow` kan se ud, fra de indledendende forberedelser til den færdige `data frame`. Undervejs skal vi se eksempler på forskellige måde at loope over indeholdet af rækker og kolonner.

Nogle af elementerne vil være repetition af elementer fra tidligere notebooks.

1. Forberedelse

Dependencies

Som altid begynder vi med at importere de nødvendige `libraries`. Vi skal bruge `os`, `Numpy` og `Pandas`, der importeres ved hjælp af `import`-kommandoen.

`np` og `pd` er blot variabelnavne. Når vi bruger `import pandas as pd`, kan vi efterfølgende nøjes med at henvise til Pandas-pakken med `pd`.

In [1]:
```
import os                       # os tillader os bl.a. finder filplaceringer på computeren
import numpy as np              # Numpy leverer noget af matematikken, der ligger under Pandas
import pandas as pd             # Pandas tillader os at importere, oprette og manipulere data frames
from pandas import DataFrame    # Nogle libraries har under-biblioteker. Underbiblioteker importeres med from-kommandoen
```

1. Forberedelse

Dependencies

Som altid begynder vi med at importere de nødvendige `libraries`. Vi skal bruge `os`, `Numpy` og `Pandas`, der importeres ved hjælp af `import`-kommandoen.

`np` og `pd` er blot variabelnavne. Når vi bruger `import pandas as pd`, kan vi efterfølgende nøjes med at henvise til Pandas-pakken med `pd`.

In [1]:
```
import os                       # os tillader os bl.a. finder filplaceringer på computeren
import numpy as np              # Numpy leverer noget af matematikken, der ligger under Pandas
import pandas as pd             # Pandas tillader os at importere, oprette og manipulere data frames
from pandas import DataFrame    # Nogle libraries har under-biblioteker. Underbiblioteker importeres med from-kommandoen
```

Definition af pipelinefunktioner

Herefter definerer vi de funktioner, vi skal bruge. Nedenfor har jeg genbrugt de to pipeline-funktioner, vi tidligere har arbejdet med. Det er en vigtig pointe at kode kan genbruges, og det er en god idé at gemme funktionalitet som funktioner. Funktioner er lette at genbruge, og I vil med tiden opbygge en samling af funktioner, der gør det let og overskueligt at lave nye analyser.

In [2]:
```
def rens_ord(text_0):
    text_1 = text_0.replace("\n"," ")
    text_2 = text_1.replace("."," ")
    text_3 = text_2.replace(","," ")
    text_4 = text_3.replace(":"," ")
    text_5 = text_4.replace("*"," ")
    text_6 = text_5.replace("-"," ")
    text_7 = text_6.replace("'"," ")
    text_8 = text_7.replace("”"," ")
    text_ren = text_8.replace("-"," ")
    text_lav = text_ren.lower()
    text_token = text_lav.split()
    return text_token
```

3. Ordfrekvens

3.a Ordfrekvens i kapitler

Vi har nu vores to `data frames` klar og kan begynde at tælle. I det følgende skal vi forsøge at skabe **overblik** over, hvor i romanen, de forskellige karakterer optræder.

Vi **definerer** først en funktion, der som **argument** tager den tekstbid vi vil søge i, og den `string` vi vil søge efter.

In [16]:
```
def find_navn(txt,navn):
    antal = txt.count(navn)
    return antal
```

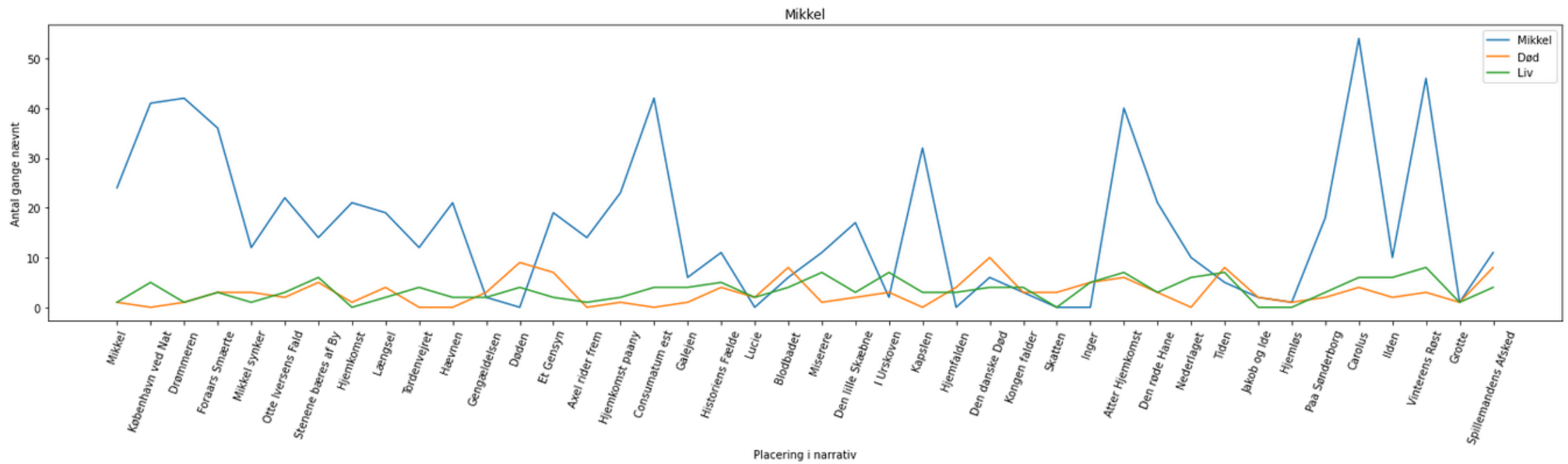
I kodesekvensen nedenfor afprøver vi funktionen på vores `data frame` og søger efter strengen **'mikkel'**. Bemærk den særlige notation `args=('mikkel',)`. Dette er nødvendigt, når vi anvender funktionen inden i `apply()`. **Husk** kommaet efter søgestrengen, det skal med selvom der kun er ét ekstra argument.

Afprøv kodesekvensen og **diskutér** hvad de enkelte dele betyder.

In [17]:
```
df_kap["antalMikkel"]  = df_kap.Kapitel_ord.apply(find_navn, args=('mikkel',))
```

In [42]:
```
kf_ordliste = rens_ord(kf_raw)
eksempel_3 = [w for w in kf_ordliste if re.search('^liv', w)]
print(len(eksempel_3))
print(eksempel_3)
```

```
59
['livløse', 'liv', 'livsaanderne', 'liv', 'livsvisdom', 'livet', 'livsdrift', 'liv', 'liv', 'liv', 'liv', 'livet', 'l
ivet', 'liv', 'livs', 'livet', 'liv', 'livssyn', 'liv', 'liv', 'liv', 'livet', 'livet', 'liv', 'liv', 'liv', 'liv', '
livet', 'liv', 'liv', 'livsdele', 'livsvarme', 'livsmod', 'livsens', 'livet', 'livet', 'livs', 'livløse', 'livligere
', 'livet', 'livet', 'livet', 'livs', 'liv', 'livet', 'liv', 'liv', 'liv', 'liv', 'livlighed', 'liv', 'livet', 'live
', 'livet', 'livet', 'liv', 'liv', 'livets', 'livtag']
```



Minieksempel på sentiment-analyse

In [4]:
```
txt = ["De", "har", "både", "en", "sportsvogn", "og", "en", "børnecontainer"]
dict = {
    "bil": 0,
    "sportsvogn": 5,
    "børnecontainer": -3,
    "skrotbunke":-5
}
score = 0

for w in txt:
    for k, v in dict.items():
        if w == k:
            score = score + v
            print("Item score:", k, v)
print("Samlet score: ", score)
```

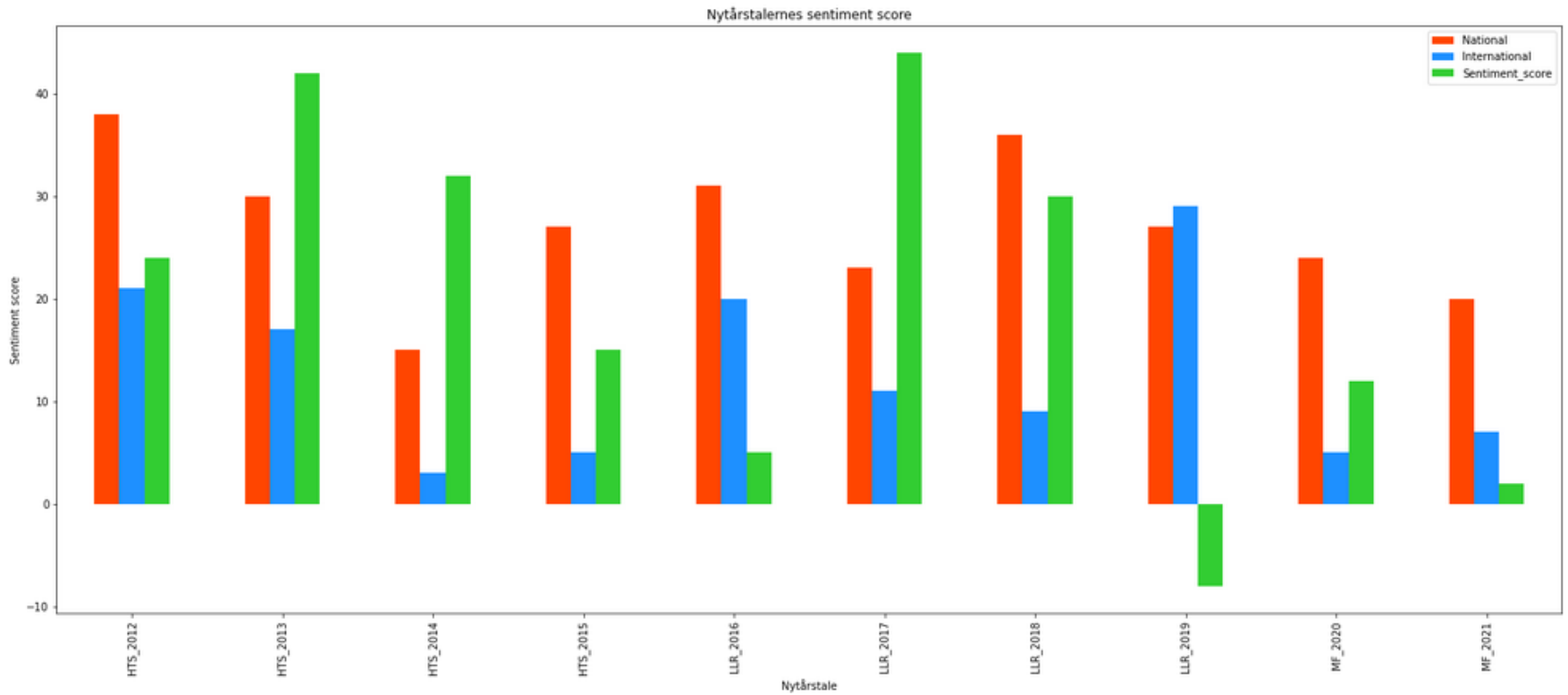
```
Item score: sportsvogn 5
Item score: børnecontainer -3
Samlet score:  2
```

Sentimentanalyse af nytårstaler

Out[45]:

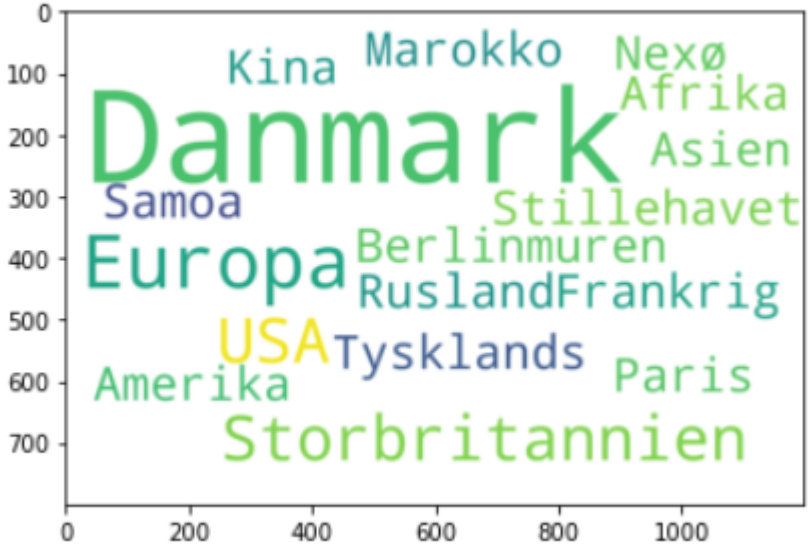
ID	National	International	Sentiment_score
HTS_2012	38	21	24.0
HTS_2013	30	17	42.0
HTS_2014	15	3	32.0

In [43]:
```
df_newYear.plot(kind="bar",figsize=(25,10), color=['orangered', 'dodgerblue', 'limegreen'])

plt.title("Nytårstalernes sentiment score")
plt.ylabel("Sentiment score")
plt.xlabel("Nytårstale")

plt.legend()
plt.show()
```



Lars Løkke Rasmussen 2019

Out[57]: <matplotlib.image.AxesImage at 0x22ebfcf34f0>



Mette Frederiksen 2021

Out[59]: <matplotlib.image.AxesImage at 0x22ebfda8940>