



# Open Data Science

## Portfolio 1, 2 & 3

Eksaminand: Martine Ingemann Jørgensen, JPR328

Gruppe:

Natacha Rylander Bech, TGZ940

Martine Ingemann Jørgensen, JPR328

Stephanie Rose Acampado Soelmark, PZG932

Antal tegn: 47 878

6 januar, 2020

## Indholdsfortegnelse

<b>PORTFOLIO 1 .....</b>	<b>3</b>
<b>Kode .....</b>	<b>6</b>
<b>PORTFOLIO 2 .....</b>	<b>14</b>
<b>Kode .....</b>	<b>16</b>
<b>PORTFOLIO 3 .....</b>	<b>24</b>
<b>Kode .....</b>	<b>33</b>
<b>ETISKE OVERVEJELSER .....</b>	<b>37</b>

## Portfolio 1

Denne opgave behandler Titanic datasættet, der indeholder data fra 887 passagerer. Herunder information om overlevelse, navn, alder med mere. Vi har analyseret denne data og i den forbindelse visualiseret resultaterne i form af tabeller. Denne rapport behandler således opgave 5 i opgavesættet.

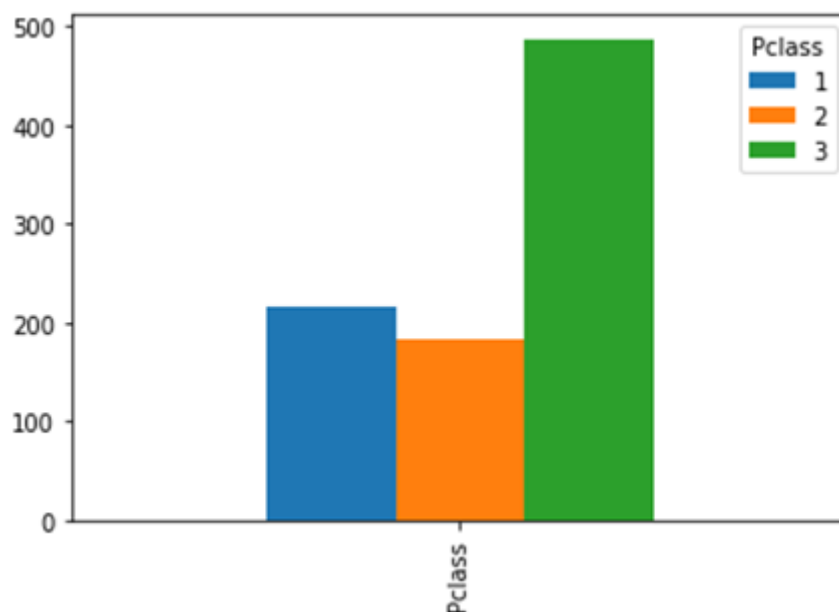
### Opgave 5 – Pivot-tabeller

Vi har først genereret en pivot-tabel, der viser antal rejsende inden for hver klasse (med brug af pandas funktioner). I dette tabel-format kan man angive hvilken kolonne, der skal vises, og hvilken funktion, der skal behandle den. Dette har vi valgt at drage fordel af i vores tabel. Dermed definerer vi columns til 'Pclass' for at få de diverse klasser og anvender aggfunc, som udgør den definition man ønsker at anvende, til at definere count, da vi ønsker en optælling af alle rejsende opdelt i klasser (pandas.pivot\_table, u.å). Herefter anvender vi plot(kind='bar'), således vi også kan få et visuelt output.

```
class_tabel = df_titanic.pivot_table(columns='Pclass', aggfunc=({'Pclass': 'count'}))
```

```
print(class_tabel)
```

```
class_tabel.plot(kind='bar')
```



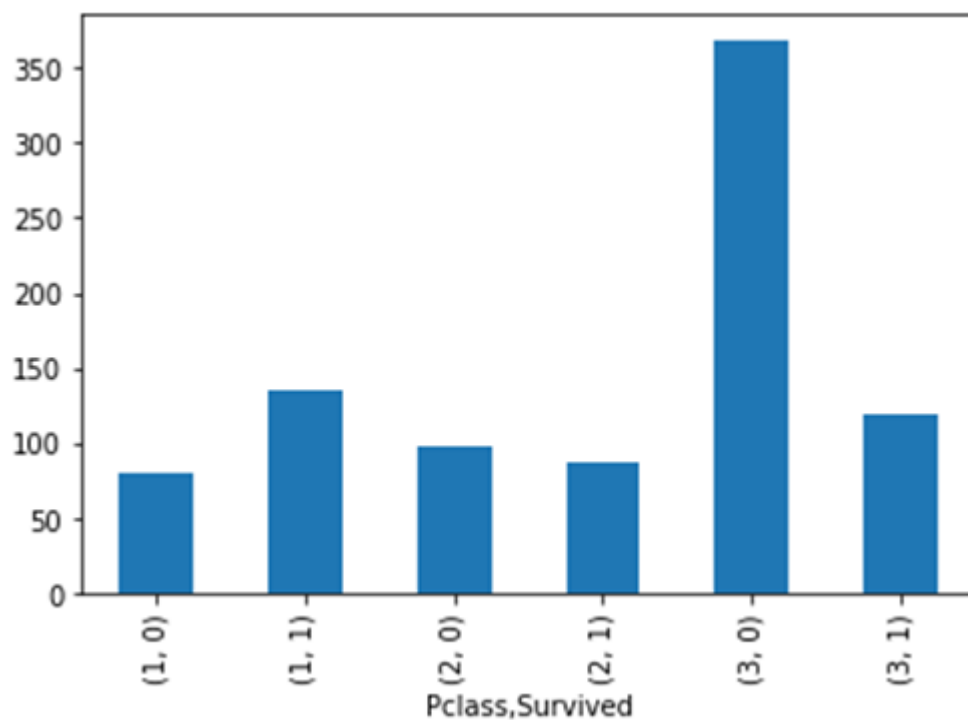
Således påvises det at der var 216 passagerer på første klasse, 184 på anden klasse og 487 på tredje klasse.

Herefter har vi lavet en pivot-tabel over antal overlevende og omkomne i hver klasse, hvor 0 er omkomne og 1 er overlevende. Funktionen optæller antallet for hver værdi i survived. Endnu engang har vi lavet en visuel fremstilling med `plot(kind='bar')` funktionen.

```
nyplot = df_titanic.groupby(['Pclass', 'Survived'])['Survived'].count()
```

```
print(nyplot)
```

```
nyplot.plot(kind='bar')
```



Således påviser søjlediagrammet at der på første klasse var 80 omkomne, på anden klasse var der 97 omkomne og på tredje klasse var der 368 omkomne. Dermed var 3 klasse den med flest omkomne

Slutteligt er endnu en pivot-tabel, der udelukkende viser fordelingen af overlevende i hver klasse. Først har vi lavet en ny dataframe, der består af kolonnerne, Survived og Class. Dernæst har vi

anvendt `.groupby()` til at visualisere fordelingen af overlevende samt omkomne på de forskellige klasser.

For at generere pivot-tabellen, anvender vi den nye dataframe, værdien af de overlevende, som bliver opstillet i en kolonne sat i forhold til de tre klasser.

For at få antallet af overlevende anvender vi `sum()`. Funktionen finder således summen af alle værdierne, bestående af 0 og 1, og fremviser disse i tabellen ud fra hver klasse. Igen anvender vi `plot(kind='bar')` for at visualisere resultaterne.

```
ny_df = df_titanic[['Survived', 'Pclass']]
```

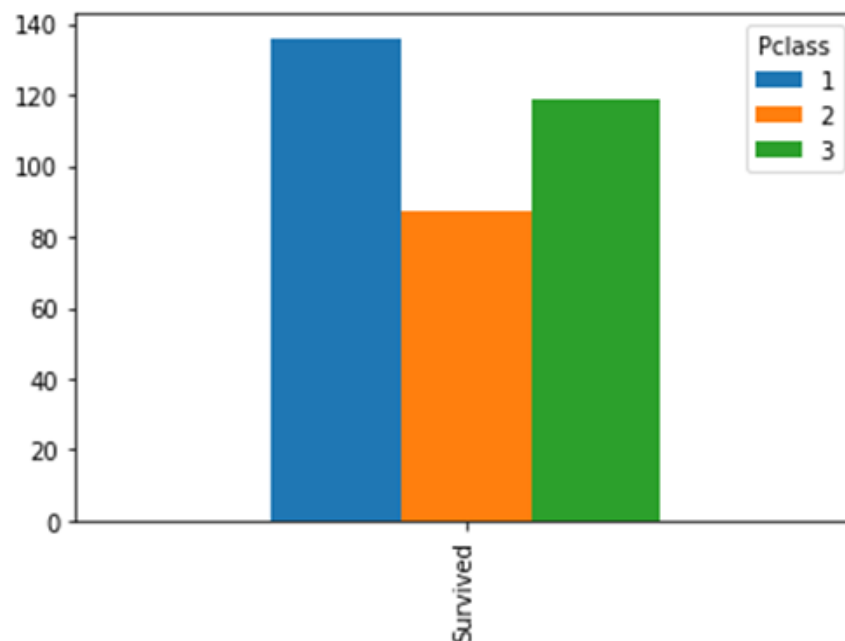
```
print(ny_df)
```

```
ny_df.shape
```

```
tabell = pd.pivot_table(ny_df, values= 'Survived', columns= 'Pclass', aggfunc= 'sum') #antal  
overlevende på de forskellige klasser
```

```
print(tabell)
```

```
vistabell = tabell.plot(kind='bar')
```



Overstående søjlediagram påviser således antal overlevende per klasse. De fleste overlevende på henholdsvis første og tredje klasse (136 og 119). Mens de færrest overlevende var på anden klasse (87).

## Litteraturliste

Pandas.pydata.org. (u.å). pandas.pivot\_table. Lokaliseret d. 19. september 2019 på:

[https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.pivot\\_table.html](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.pivot_table.html)

## Kode

```
# OPGAVE (1) Åben filen i en tekst-editor og se på indholdet.
```

```
# OPGAVE (a)
```

```
# I filen 'titanic.csv', som er åbnet med Numbers, kan man anskue følgende datatyper, 'interger' og 'float', 'string'.
```

```
# Den førstnævnte angiver, at værdien er et heltal, den næste et decimaltal og sidstnævnte et tekststykke.
```

```
# Endvidere er hele datasættet opdelt i 8 kolonner: 'Survived', 'Pclass', 'Name', 'Sex', 'Siblings/Spouses Aboard', 'Parents/Children Aboard' og 'Fare'.
```

```
# De overlevende/omkomne bliver angivet som henholdsvis '1'/'0' (intergers).
```

```
# Dernæst kan man også se opdelingen af klasser, 1, 2 og 3 (interger), samt navne på passagerne og deres køn (strings).
```

```
# Derudover kan man også se deres alder, antal søskende, ægtefæller og forældre (interger) og billetprisen (float).
```

```
# OPGAVE (b)
```

```
# Når vi gennemgår Numbers-arket, kan vi se, at der ikke mangler data, eftersom der ikke forekommer tomme celler i dataframen.
```

# Manglende værdier vil fremstå som tomme celler i Numbers og kategoriseret som 'NaN' (Not a Number) i Python.

# Dette demonstreres også i næste opgave.

# OPGAVE (2)

# Her importeres pandas

```
import pandas as pd
```

# Her læser vi filen titanic med read\_csv, da det er en csv-fil.

```
df_titanic = pd.read_csv('titanic.csv')
```

# Dernæst udskriver vi dataframen og kan se, at der i alt eksisterer 887 rækker og 8 kolonner, hvilket svarer til 887 passagerer og 8 kategorier.

```
print(df_titanic)
```

# Nu vil vi bruge describe() til at få et hurtigt overblik over diverse kolonner og generelle statistiske beregninger.

# Her kan vi eksempelvis se, at den højeste pris for en billet er 512,8 og den mindste pris er 0.

```
df_titanic.describe()
```

# klart overblik over alle 8 kategorier i for-loop, der gennemløber dataframen og udskriver kategorien for hvert gennemløb.

```
for i in df_titanic.columns:
```

```
    print(i)
```

# Her bruger vi shape() til at illustrere antallet af kolonner og rækker i dataframen.

```
print(df_titanic.shape)
```

# Denne funktion viser antallet af celler. Således kan man se antallet af værdier i hele dataframen

```
print(df_titanic.size)
```

```
# viser datatyper. Funktionen undersøger og definerer forskellige typer data i dataframen og fortæller, at der findes følgende: Int64, object, float64.
```

```
# Dette kan defineres som henholdsvis integer numbers, string, og float.
```

```
# Pandas referer til strings som object. (Kilde:
```

```
https://pbpython.com/pandas\_dtypes.html?fbclid=IwAR06-
```

```
ND1itRz3rV4UNzihOOLj5IgtVpgZm2Z6FDVsdyVCe4UQhf7jgLb--Y\)
```

```
print(df_titanic.dtypes)
```

```
# Undersøger om dataframen mangler data. Funktionen stiller spørgsmålet: Er der tomme celler i datasættet. Hvis nej, vil output være False og hvis ja, vil den returnere True.
```

```
df_titanic.empty
```

```
# Alternativ løsning til at finde missing data. Bruger funktionen isnull() og sum() til at finde missing data. Funktionen gennemgår data for hver kolonne, og optæller antallet af True og False, da den ligeledes forsøger at undersøge, om værdien er 0.
```

```
# Såfremt funktionen finder tomme celler, vil den angive dem som True og lave en optælling af dette booleanske udtryk. Hvis funktionen ikke finder tomme celler, vil de blive angivet som False og tilskrevet værdien 0.
```

```
# Derved får vi en værdi ud fra hver kolonne som output, der viser tallet 0, eftersom der ikke forekommer tomme celler i dataframen.
```

```
print(df_titanic.isnull().sum())
```

```
# OPGAVE (3)
```

```
# Denne funktion udskriver antallet af overlevende, da den lægger summen af alle værdierne sammen.
```

```
# 0 er omkomne og 1 er overlevende.
```

```
# Således ved vi med sikkerhed at tallet, 342, er antallet af overlevende og er kategoriseret som tallet 1.
```



```
print(df_titanic['Survived'].sum())
```

```
# Denne funktion tæller antallet af hver unikke værdi i kolonnen.
```

```
# Her kan vi se både antallet af omkomne (0), 545, og overlevende (1), 342.
```

```
print(df_titanic['Survived'].value_counts())
```

```
# Overblik over passasgerne på hver klasse. Vi har brugt samme funktion, der tæller antallet af hver unikke værdi.
```

```
print(df_titanic['Pclass'].value_counts())
```

```
# Alternativ måde at se antal rejsende inden for hver klasse med en variabel.
```

```
filter_classes=df_titanic['Pclass'].value_counts()
```

```
print(filter_classes)
```

```
# Gennemsnitsalder med mean()funktionen.
```

```
# Vi angiver en specifik kolonne i dataframen og bruger mean() funktionen.
```

```
average= df_titanic['Age'].mean()
```

```
# Her runder vi tallet til nærmeste heltal.
```

```
print(round(average))
```

```
# Vi bruger median(), således vi kan se median alderen.
```

```
print(df_titanic['Age'].median())
```

```
# Vi udskriver kolonnen Name, så vi kan få en oversigt over navnene.
```

```
print(df_titanic['Name'])
```

```
# Her kan vi få et overblik over den højeste billetpris ved hjælp af max() funktionen.
```

```
print(df_titanic['Fare'].max())
```

```
# Her vil output vise den laveste billetpris ved hjælp af min() funktionen.
```

```
print(df_titanic['Fare'].min())
```

```
# Her kan vi se gennemsnitsprisen på billetterne ved at bruge mean() igen.
```

```
print(df_titanic['Fare'].mean())
```

```
# OPGAVE (4)
```

```
# For at undersøge, hvorvidt der forekommer tilfælde, hvor passagererne har samme efternavn, har vi først og fremmest kreeret en ny variabel og dernæst benyttet .str.split() funktionen.
```

```
# Denne funktion splitter kolonnen, Name, i et antal kolonner og opdelinger i forhold til en bestemt seperator.
```

```
# Først kan man definere, hvorvidt kolonnen skal opdeles efter en streng eller regulært udtryk i Pat.
```

```
# Dernæst hvor mange opdelinger, der skal forekomme i N, eg. Hvis n = 1, får man 2 opdelinger af dataen, således et navn freemstår som en liste med to elementer: forenavn og efternavn.
```

```
# Slutteligt kan man vælge i expand, om strengene skal udvides i separate kolonner baseret på booleanske udtryk, True og False.
```

```
# Hvis man ikke definerer Pat, vil funktionen, som standard, separere efter mellemrum, hvilket passer med separeringen af navnene.
```

```
# Vi har valgt kun at inkludere N og expand. N er sat til 1 og expand er True. Således får vi to kolonner med opdelingen, fornavne og efternavne.
```

```
# Slutteligt har vi udskrevet antal efternavne (kolonne 1, da kolonne 0 er fornavne) i den nye variabel sammenlagt med antallet af gentagelser, da value_counts tæller de unikke værdiers forekomst.
```

```
# kilde: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.str.split.html
```

```
df_new= df_titanic['Name'].str.split(n=1, expand=True) #kolonner med efternavne
```

```
print (df_new)
```

```
df_new[1].value_counts() #navne + antal gentagelser
```

```
#En alternativ løsning på opgaven viser alle de efternavne, der går igen i dataframen.
```

```
#Først har vi lavet en variabel, som finder Names og splitter dem ved mellemrum bagfra.
```

```
lastnames=df_titanic.Name.str.split(' ').str[-1]
```

```
print(lastnames)

# Her kan vi se hele listen over alle efternavnene, alle 887.
print(lastnames.tolist())

# Her har vi oprettet to tomme liste variable, som vi ønsker at bruge senere.
unique=[]
repeat=[]

# Vi har lavet en for-loop, der tager udgangspunkt i listen med alle efternavnene
# Ved første for-loop runde leder funktionen efter et efternavn.
# Hvis efternavnet ikke findes i 'unique' listen i forvejen, bliver det tilføjet til den tomme unique
liste.
# Ved anden for-loop runde, leder funktionen også først efter om efternavnet findes i 'unique' listen
- og hvis det ikke gør, er det stadig unikt og skal tilføjes til den tomme liste
# Derimod hvis efternavnet allerede eksisterer i 'unique', går efternavnet videre til elif-statementet
og spørger om efternavnet findes i repeat-listen - på samme måde tilføjes det til repeat listen, hvis
navnet ikke er der i forvejen
# Resultatet ender ud med, at alle navne i repeat-listen kun bliver nævnt 1 gang - og vi kan derfor
konkludere, at 133 navne ud af de ialt 887, bliver gentaget mere end 1 gang

for i in lastnames:
    if i not in unique:
        unique.append(i)
    elif i not in repeat:
        repeat.append(i)

print(sorted(repeat))
print(len(repeat))

# OPGAVE (5)
```

```
# Først har vi genereret en pivot-tabel med klasse, der viser antal rejsende inden for hver klasse
(med brug af pandas funktioner.)
# I dette tabel-format kan man angive hvilken kolonne, der skal vises, og hvilken funktion, der skal
behandle den. Dette har vi valgt at drage fordel af i vores tabel.
# aggfunc udgør den type funktion, man ønsker at anvende, hvilket vi har defineret som count, da vi
skal have en optælling af alle rejsende opdelt i klasser.
# Kilde: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.pivot\_table.html
class_tabel= df_titanic.pivot_table(columns='Pclass', aggfunc=({'Pclass':'count'}))
print(class_tabel)
#Her bruger vi plot(kind='bar'), således vi også kan få et visuelt output
class_tabel.plot(kind='bar')

#Antal overlevende/omkomne i hver klasse, hvor 0 er omkomne og 1 er overlevende. Funktionen
optæller antallet for hver værdi i survived.
# I første klasse var der 80 omkomne
# I anden klasse 97
# I tredje klasse 368
# Dermed var tredje klasse den med flest omkomne.
nyplot = df_titanic.groupby(['Pclass', 'Survived'])['Survived'].count()
print(nyplot)
# Her har vi lavet en visuel fremstilling af pivot-tabellen.
nyplot.plot(kind='bar')

# Her har vi lavet en Pivot tabel, der udelukkende viser fordelingen af overlevende i hver klasse.
# Først har vi lavet en ny dataframe, der består af kolonnerne, Survived og Class. Dernæst har vi
brugt .groupby() til at visualisere fordelingen af overlevende samt omkomne på de forskellige
klasser.
# For at generere pivot-tabellet, vil vi anvende den nye dataframe, værdien af de overlevende, som
bliver opstillet i kolonne i forhold til de tre klasser.
# Denne pivot-tabel bruger sum(), således vi kan se antallet af overlevende.
# Funktionen finder summen af alle værdierne, bestående af 0 og 1, og fremviser dem tabellen ud
fra hver klasse.
```

# Slutteligt har vi inkluderet plot(kind='bar'), for at demonstrere fordelingen i et søjlediagram.

```
ny_df = df_titanic[['Survived','Pclass']]
print(ny_df)
ny_df.shape
tabel1= pd.pivot_table(ny_df, values= 'Survived', columns= 'Pclass', aggfunc= 'sum') #antal
overlevende på de forskellige klasser
print(tabel1)
# Her har vi lavet en visuel fremstilling af pivot-tabellen.
vistabel1 = tabel1.plot(kind='bar')
```

# Det sidste vi vil demonstrere er, hvor mange der i alt overlevede og omkom, ved brug af en for-loop.

# Den første funktion laver en liste over alle 1'ere og 0'ere - som skal bruges i for-loop funktionen.

```
count_people=df_titanic['Survived'].tolist()
print(count_people)
```

#Denne for-loop løber derfor igennem listen 'count\_people', og tæller hvor mange der overlevede og hvor mange der døde

```
survived=0
not_survived=0
for i in count_people:
    if i == 0:
        not_survived = not_survived + 1
    elif i == 1:
        survived = survived + 1
print(survived)
print(not_survived)
```

## **Portfolio 2**

### **Introduction**

For this assignment we will be working with the “The Guardian” dataset. To define a research question, we looked through the dataset and found it to be news articles. We then limited the data, meaning the articles, to a month, here from September 1 to November 1, 2019. From this we got 1669 articles. In these we looked for repetitions and found that Boris Johnson’s name came up several times. This led us to the following research question:

Under which circumstances were 'Boris Johnson' discussed in relation to the 'Politics' category, in the Guardian from September 1 to November 1, 2019.

### **Our key findings**

#### **Task 1**

For this task we printed the number of id's/text's/fields' which are 1669, symbolizing the number articles in our chosen dataframe.

#### **Task 2**

To derive a document matrix, we used CountVectorizer to remove stopwords, added a token pattern with a regular expression and subsequently transformed the data to a matrix in our “vecfit” variable. Then, we can illustrate the amount of unique words that were tokenized, namely, 43713 words from our text corpus.

We calculate the sparsity of our matrix to be 0.99, which signifies that our matrix is sparse. As the next step, we calculate the TF-IDF weighting of our matrix.

A key finding in this task is the previous and the tokenized word count. The previous is 2.330.313, while the tokenized is 1.055.035. We have used CountVectorizer dictionary stopwords and the regular expression for this.

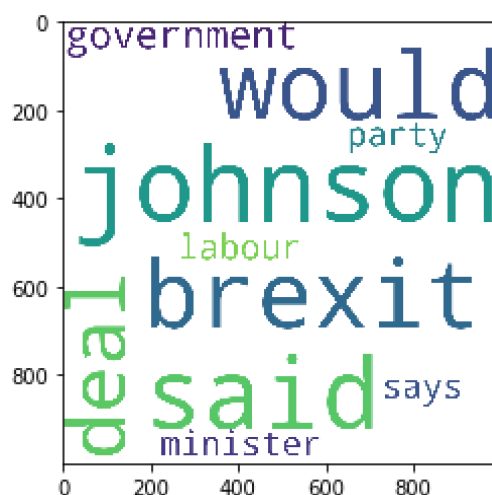
Another key finding is the average length of a document. We calculated this to be 632,1 after removing the stopwords from the text.

### Task 3

For this task we narrowed down our search and generated a query, focusing on 'Politics' in relation to Boris Johnson. We choose this topic in order to answer our research question. Hereafter, we created a matrix over our query to use for the next task.

### Task 4

In relation to topic modelling, we made the following word cloud. This illustrates the top 10 words in four different components. Thus, it becomes evident that topics regarding brexit, government and the deal have been on the agenda during the two months, September and November. These comply with reality as they do in fact reflect current and well-debated topics in the UK. Therefore, by further assessing and interpreting the situation in the UK based on news, we can use our model to identify and reveal interesting topics that may set the scene for further research.



### Conclusion

To answer our research question, from our topic modelling we can conclude that Boris Johnson is discussed in relation to politics, Brexit, the Labour party, government, minister and England. These topics makes sense as Boris Johnson is the newly appointed prime minister of England, and thereby a part of the government and the ongoing discussion of Brexit.

## Kode

```
# Imports
## Task 1
import json
import os
import pandas
import requests
from os import makedirs
from os.path import join, exists
from datetime import date, timedelta

## Task 2
import pandas as pd
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from nltk.probability import FreqDist

## Task 4
from sklearn.decomposition import LatentDirichletAllocation
from wordcloud import WordCloud
import matplotlib.pyplot as plt

''' Task 1 '''

# Here we download the data

ARTICLES_DIR = join('theguardian', 'collection')
```



```
makedirs(ARTICLES_DIR, exist_ok=True)
MY_API_KEY = 'bfeb3d66-1e24-4e0a-8082-1c6ee9e2a7b7'

API_ENDPOINT = 'http://content.guardianapis.com/search'
# We create a dictionary called my_'params' and set the values
# Boris Johnson will be our focus. Thus, we will import articles in relation to him.
# for the API's mandatory parameters
my_params = {
    'q': 'boris+johnson',
    'from-date': '',
    'to-date': '',
    'order-by': "newest",
    'show-fields': 'all',
    'page-size': 200,
    'api-key': MY_API_KEY
}

# We choose to download the data with this start- and end date.
# We have chosen to download only one week of data to narrow down
# the text corpus.
start_date = date(2019, 9, 1)
end_date = date(2019, 11, 1)

dayrange = range((end_date - start_date).days + 1)
for daycount in dayrange:
    dt = start_date + timedelta(days=daycount)
    datestr = dt.strftime('%Y-%m-%d')
    fname = join(ARTICLES_DIR, datestr + '.json')
    if not exists(fname):
        print("Downloading", datestr)
        all_results = []
        my_params['from-date'] = datestr
```

```
my_params['to-date'] = datestr
current_page = 1
total_pages = 1
while current_page <= total_pages:
    print("...page", current_page)
    my_params['page'] = current_page
    resp = requests.get(API_ENDPOINT, my_params)
    data = resp.json()
    all_results.extend(data['response']['results'])
    current_page += 1
    total_pages = data['response']['pages']
with open(fname, 'w') as f:
    print("Writing to", fname)
    f.write(json.dumps(all_results, indent=2))
```

# Here we read the data

```
directory_name = "theguardian/collection/"
```

```
ids = list()
```

```
strtexts = ""
```

```
texts = list()
```

```
allfields = list()
```

```
allheadlines = list()
```

```
allSections = list()
```

for filename in os.listdir(directory\_name): #For ever file in the guardian collection, listdir every file in this collection - it returns an unsorted list of all the directories in the path..

if filename.endswith(".json"): # If the file ends with json, open the collection as well as filenames as json files.

```
with open(directory_name + filename) as json_file:
```

```
data = json.load(json_file) #saves the json-files in a variable, names data.
for article in data: #for every article in the variable data.
    id = article['id'] # id is defined as the articles ids
    headline = article['webTitle'] #the variable headline is defined as the html-title, 'webTitle',
in the corpus.
    sections = article['sectionName']# etc.
    fields = article['fields']
    text = fields['bodyText'] if fields['bodyText'] else ""
    ids.append(id) #appends the variable id to a list
    strtexts += text #adds text to the variable to create a string version
    texts.append(text) # creates a list version of all the texts
    allSections.append(sections)
    allfields.append(fields)
    allheadlines.append(headline)

print("Number of ids:", len(ids))
print("Number of texts:", len(texts), "Number of fields", len(allfields))

""" Task 2 """

# We have generated a dataframe for a better overview of the dataset

dataframe = pd.DataFrame({'all_fields': allfields,
                           'all_headlines': allheadlines,
                           'all_texts': texts,
                           'section_ids': allSections})
dataframe.shape # (13713 rows and 5 columns)

# We will derive a document-term matrix for our collection.
# We remove stopwords and count the amount of words,
```

# and lastly convert the data to a matrix.

```
countvect = CountVectorizer(min_df = 1,  
                             stop_words = stopwords.words('english'),  
                             token_pattern = r'[a-zA-Z\-\_][a-zA-Z\-\_]{2,}')  
vecfit = countvect.fit_transform(texts) # transforming data to matrix, vectors.
```

# We print a list over terms from the variable, texts, sorted according to their index, which is produced by fit\_transform

```
matrixwords = countvect.get_feature_names()  
print(matrixwords)  
matrixxx = vecfit.toarray()  
print(matrixxx)  
# Here we print the shape of our matrix  
print(vecfit.shape) #1669, 43713
```

# Show words and index numbers

```
# Access the entire vocabulary to see what exactly was tokenized by calling  
top_ordindex=countvect.vocabulary_  
print(top_ordindex)
```

# Amount of words in new vocabulary - only unique words

```
print(len(countvect.vocabulary_))
```

# Calculating the matrix sparsity

```
sparsity = 1.0- np.count_nonzero(matrixxx) / matrixxx.size  
print(sparsity)
```

# Result: 0.99.

# Thus, our matrix is sparse as its sparsity is greater than 0.5.

```
# TF-IDF weighting.
# Calculating the weight of the words by using scikit's tfidftransform
# on our previous document-term count matrix.
tfidfmodel = TfidfTransformer()
datafittransformer = tfidfmodel.fit_transform(vecfit)
print(datafittransformer.shape)
print(datafittransformer.toarray())

#How many documents = 1669
len(texts)

# Pre-processing
# Word count before and after pre-processing
# Tokenize loop

# We ensure that the text (as a string) is divided into words by using word_tokenize
tokenizedtext = word_tokenize(strtexts)
no_stopwords = []

# Then, we save the stopwords in a variable
stop_words = set(stopwords.words('english'))

# And create a loop, that goes thorough all the words in the tokenized text and checks if it contains
stopwords. If not, it will be saved to a list.
for bestemtord in tokenizedtext:
    if bestemtord.lower() not in stop_words:
        if bestemtord.isalpha():
            no_stopwords.append(bestemtord)

print("Previous word count: ", len(tokenizedtext)) #2330313
```

```
print("Word count (removed stopwords and tokens): ", len(no_stopwords)) #1055036
```

```
# Average length of documents = 632.1 words - after tokenization
```

```
len(no_stopwords)/len(texts)
```

```
# Top 20 most common words in our collection. 'Johnson' and 'Brexit' are very popular words.
```

```
ferquencywords = FreqDist(no_stopwords)
```

```
top_topwords = ferquencywords.most_common(20)
```

```
print(len(top_topwords)) # gets total amount of topics
```

```
print(top_topwords)
```

```
# Distribution of articles in different topics
```

```
ferquencynewsgroups = FreqDist(dataframe['section_ids'])
```

```
top_newsgroups = ferquencynewsgroups.most_common(30)
```

```
print(len(top_newsgroups)) # gets total amount of topics
```

```
print(top_newsgroups)
```

```
""" Task 3 """
```

```
#First, we create a query focusing on politics to explore the current agenda in relation to Boris Johnson.
```

```
terms = ['Politics']
```

```
terms
```

```
query = " ".join(terms)
```

```
query
```

```
# Then we create a matrix of our query, countvect
```

```
query_vect_counts = countvect.transform([query])
```

```
query_vect = tfidfmodel.transform(query_vect_counts)
```

query\_vect

""" Task 4 """

# We generated a topicmodel with one component since we are only focusing on politics in relation to Boris Johnson (whom we specified within our search parameters in the beginning)

# and make it replicable to ensure that the results will be the same each time we run the code.

topicModel\_lda = LatentDirichletAllocation(n\_components=1, random\_state=0)

# We use our previous document-term count matrix, vecfit.

data\_lda = topicModel\_lda.fit\_transform(vecfit)

np.shape(data\_lda)

print(data\_lda)

#We sort the term weights according to the 10 most popular words in a loop.

#then we get the topwords from countvect.get\_feature\_names() to show the words.

for i, term\_weights in enumerate(topicModel\_lda.components\_):

top\_idx = (-term\_weights).argsort()[:10]

top\_words = ["%s (%.3f)" % (countvect.get\_feature\_names()[idx], term\_weights[idx]) for idx in top\_idx]

print("Topic %d: %s" % (i, ".join(top\_words)))

#We included a wordcloud to present our results.

#This involved creating a new variable that contained a dictionary of topwords and termweights.

#Then our wordcloud can show the most popular words according to their weight.

for i, term\_weights in enumerate(topicModel\_lda.components\_):

top\_idx = (-term\_weights).argsort()[:10]

top\_words = [countvect.get\_feature\_names()[idx] for idx in top\_idx]

word\_freqs = dict(zip(top\_words, term\_weights[top\_idx]))

wc = WordCloud(background\_color="white",width=1000,height=1000,  
max\_words=10).generate\_from\_frequencies(word\_freqs)

plt.subplot(1, 1, i+1)

plt.imshow(wc)

# Thus, it becomes evident that topics regarding brexit, government and deal have on the agenda during these last months.

# These comply with reality as they are in fact current and well-debated topics in the UK.

## Portfolio 3

### Beskrivelse af SMK Open

Vores valgte institution er Statens Museum for Kunst (SMK), da denne på nuværende tidspunkt fører et projekt der hedder SMK Open (2016-2020). Dette går ud på "...at stille hele Danmarks kunstsamling til fri afbenyttelse." (SMK Open, u.å.) ved at digitalisere og tilgængeliggøre museets samling. SMK's formål er at alle danskere skal kunne benytte sig af kunsten og anvende den i "...sit eget liv og bruge på sine egne vilkår" (SMK Open, u.å.). Dermed har SMK tilgængeliggjort alt deres data ved at lade interessenter benytte SMK's egne API (SMK's API (beta-version), u.å.). API (Application Programming Interface) er en tjeneste der muliggør at software kan tale med andet software. Det er med til at digitale tjenester kan udgive og offentliggøre deres data for andre interessenter på en struktureret måde (SMK's API (beta-version), u.å.).

SMK forestiller sig eksempelvis at deres data kan; tilgås uafhængigt af tid og rum; viderebearbejdes; nærstuderes i detaljer; deles; indsættes i alt fra bøger til forskningsartikler og skoleopgaver; og trykkes på alt fra plakater til sofapuder (SMK Open, u.å.).

Et konkret eksempel på anvendelse af SMK's data er applikationen Vizgu. Dette er en applikation der i samarbejde med diverse museer, heriblandt SMK, gør data tilgængelig for individet, mere specifikt er det en digital guide. Applikationen fungerer sådan at man blot skal pege sin telefon mod et kunstværk, hvorefter man vil få en masse yderligere oplysninger om det pågældende værk (vizgu, 2019; SMK Vizgu, u.å.).

Endvidere har SMK i samarbejde med Hack4DK udforsket mulighederne ved de digitaliserede kunstværkerne med projekter, der trækker på kulturarvsdata (SMK Fridays: Deler kunsten, u.å.). Et eksempel på et projekt fra dette event indbefatter et spil, der tilskynder det enkelte individ at gætte titlerne på diverse malerier ud fra en 3D illustration (hack4dk, u.å.).



Således har SMK efterlevet deres mål om at tilgængeliggøre deres digitaliserede værker for offentligheden og dermed også opfordre samfundet til at anvende denne form for data til teknologiske og innovative måder at videregive og formidle kulturarven.

## Opgave 2

### Generering af URL

Vi har genereret vores URL, der søger efter statuetter i SMK's digitale database, som er følgende:

<https://api.smk.dk/api/v1/art/search?keys=statuette&rows=1000&encoding=json>

Denne har vi indsat i JSON Beautifier, for at kunne beskrive indholdet.

### a) Typer af metadata

Vi har identificeret forskellige slags metadata, som vi har kategoriseret i forhold til deskriptiv, administrativ og strukturel metadata.

De deskriptive metadata, bruges til at kunne identificere og beskrive indholdet på et overordnet plan, uden at gå i dybden med at forklare og vurdere (Digital Bevaring, u.å.).

De administrative metadata repræsenterer informationer om tid/dato, digitalisering og tekniske informationer og rettigheder (Digital Bevaring, u.å.).

De strukturelle metadata bruges til henholdsvis at vise og navigere samt kan det også være information om den interne organisering eller en rækkefølge af dokumenter (Digital Bevaring, u.å.).

Med henblik på førstnævnte, deskriptiv data, har vi eksempelvis observeret benævnelsen af årstal på værket: 'period', materiale: 'material', teknik: 'technique', kunstner: 'creator', fødselsår:

'creator\_date\_of\_birth', dødsår: 'creator\_date\_of\_birth', nationalitet: 'creator\_nationality', titel: 'titles', noter: 'frame\_notes' og 'content\_notes', farver: 'colors', samt beskrivelse: 'content\_description'.

Dernæst har vi i relation til administrativ metadata fundet id-numre: 'id', referencenumre: 'object number', digitaliseringsdage: 'created', rettigheder: 'public\_domain', 'copyright', afdeling på museet: 'responsibledepartment', ændringsdato: 'modified', udstilling: 'exhibition', placering: 'shelfmark'.

Slutteligt har vi udpeget de strukturelle metadata i vores dataframe, bestående af antal dele: 'parts', kollektion: 'collection', dimensioner: 'dimensions' og hvad kunstværket er en del af 'parts of'.

## **b) Sammenligning af metadata med Dublin Core**

Nedenstående er en oversigt over 15 metadata kerneelementer defineret af Dublin Core, som vi har sammenlignet med vores SMK datasæt og dermed kommet med eksempler på. Følgende 15 kerneelementer er hentet fra Dublin Cores hjemmeside (Dublin Core, 2019):

Contributor – “An entity responsible for making contributions to the resource.”

- På trods af at vi har kendskab til at SMK som organisation står for at tilbyde billeder, står dette ikke beskrevet i deres metadata.

Coverage – “The spatial or temporal topic of the resource, the spatial applicability of the resource, or the jurisdiction under which the resource is relevant.”

- Spatial: Vi har fundet lokationen på værket i form af: ‘shelfmark : 60875’.
- Temporal: I metadataen kan vi finde både digitaliseringsdag: ‘created’, ændringsdato: ‘modified’, start og slutdato på perioden: ‘start’, ‘end’, ‘period’, samt årstal på kunstværket: ‘acquisition\_date\_precision’.
- Jurisdiction: Selve afdelingen som værket hører ind under, er også nævnt som ‘responsible\_department’.

Creator – “An entity primarily responsible for making the resource.”

- Kunstneren er nævnt under produktionsdetaljer om værket som ‘creator’.

Date – “A point or period of time associated with an event in the lifecycle of the resource.”

- Perioden som startdato: ‘start’ og slutdato: ‘end’ på perioden kan man finde i metadataen under objektet ‘production\_date’.

Description – “An account of the resource.”

- Der findes diverse beskrivelser af værket i metadaten, eksempelvis: noter til indholdet: ‘notes’, indholdsbeskrivelse: ‘content\_description’, noter til opsætningen: ‘frame\_notes’, dimensioner: ‘dimensions’, og dokumentation: ‘documentation’.

Format – “The file format, physical medium, or dimensions of the resource.”

- SMK har inkluderet dimensionerne på deres værker, der indbefatter højde og centimeter under 'dimensions'.

Identifier – “An unambiguous reference to the resource within a given context.”

- Her har SMK inkluderet ID numre: 'ID', referencenumre: 'object number', lokation: 'shelfmark' og 'current\_location\_name'.

Language – “A language of the resource.”

- Derudover er forskellige sprogversioner af værkernes titler inkluderet i datasættet under elementet 'titles'.

Publisher – “An entity responsible for making the resource available.”

- Vi fandt et element i metadataen under “notes”, der står beskrevet som forskellige kataloger og hæfter - hvilket muligvis kan være dem, der er ansvarlige for at gøre statuetterne tilgængelige/dem der har givet SMK disse værker.

Relation – “A related resource.”

- Der er metadata omkring relaterede værker i form af en note: 'notes' omkring eksempelvis kataloger, som værket indgår i, derudover står der også information om samlingen.

Rights – “Information about rights held in and over the resource.”

- Vi fandt metadata, der angiver rettighederne omkring værket. Herunder er det synliggjort, hvilken form for kreditering værket er registreret med. Eksempelvis ift. copyright kreditering: 'public domæne', og rettigheder: "rights", hvor værdien kan være creative commons.

Source – “A related resource from which the described resource is derived.”

- Kilden er beskrevet ved nogle af dem som "source", hvor de har tilhørende værdier som "THL", eller "Eva De La Fuente Pedersen".
- Under notering: 'notes' der står også ved specifikke værker, hvorfra de er erhvervet. Eksempel fra 'notes' er: 'Erhvervet af Herbert Melbye den 24. juni 1942 på auktion hos Winkel & Magnussen. kat. 54 (for DKK 517,50). Bruun Rasmussens bogfortegnelse nr. 35.'

Subject – “The topic of the resource.”

- Kollektionen, udstillingen som værket er del af, indenunder ‘content\_description’, får man nogle nøgleord og sætninger omkring værket, ‘text’ der beskriver værket.

Title – “A name given to the resource.”

- Titlen på værket er ligeledes til stedet samt forskellige sproglige versioner af den, under ‘title’.

Type – “The nature or genre of the resource.”

- Dette er beskrevet under “titles”, og herunder “type”, hvoraf det kan være “Museum”.

### **c) Statistiske beregninger**

I dataen er værdier man kan anvende til statiske beregninger. Man kan blandt andet regne på årstal herunder digitaliseringsdato og produktionsdato. Med disse kan man eksempelvis lave en statistik over antal værker produceret i et bestemt år, eller en oversigt over værker digitaliseret på bestemte datoer. En anden værdi er periode, denne kan man eksempelvis anvende til at undersøge hyppigheden af perioder på SMK. Derudover kan man lave statistik over teknikker og materialer, eksempelvis hvor hyppigt et bestemt materiale eller en bestemt teknik er anvendt. Kunstnere i datasættet er også relevante at anvende, her kan man undersøge, hvor mange værker en bestemt kunstner har udstillet på SMK, eller hvilke materialer en bestemt kunstner anvender sig af hyppigst.

### **d) Eksempler på relevante spørgsmål til analyse af omtalte datasæt**

- I hvilket årstal producerede man flest statuetter? (i vores sample bestående af 1000 værker)
  - Hvad er frekvensdistributionen af statuetterne inden for hvert årstal?
- Hvor mange unikke kunstnere har skabt disse statuetter og med hvilket materialer?
- Hvordan er fordelingen af kunstnere der stadig lever og ikke lever?
- Hvordan er fordelingen af kunstnernes nationaliteter i vores datasæt? og hvilken nationalitet er hyppigst?

- Hvilken periode er den mest hyppigste inden for skulpturering?

### Opgave 3

#### a) Importering, beskrivelse og rensning af datasæt

Vores datasæt består af 537 rækker og 49 kolonner før vi har rensset det. Herunder er der adskillige kolonner, hvis værdier betegnes som 'NaN' not a number, da der mangler data - eksempelvis ved de kolonner, der starter med 'image\_'. Efter rensningen af dataframen har vi nu 537 rækker og 26 kolonner. Det vil sige at der er lige så mange statuetter som før, men med  $(49-26) = 23$  færre kolonner (elementer/typer data). Hovedparten af de kolonner vi har fjernet, er dem, hvis værdier står som 'NaN', såsom billeder eller billedinformationer. Vi vil gerne fokusere på kolonner, der er mere konkrete og kan være behjælpelige i statistiske analyser, eksempelvis; kunstner, produktionsår, nationalitet, materialer.

I vores rensede datasæt er datatyperne 'objects', 'floats' og 'booleans'. Der 537 rækker, samt 26 kolonner. Vi har fravalgt visse kolonner da disse ikke er relevante for os, eksempler er; 'iiif\_manifest', 'object\_history\_note', 'image\_native' og 'image\_thumbnail'.

Se bilag 1 for tabeller over rensning af datasættet.

### Opgave 4

#### Beregning af udvalgt data

Vi kan lave en optælling af årstallene for at undersøge, hvornår der blev fremstillet flest statuetter. Samme metode kan benyttes med henblik på typer, hvor vi kunne observere at kategorier såsom statuetter, tegninger, skulpturer indgik i vores dataframe. Vi har specificeret statuette som nøgleord i vores søgning, hvilket ikke udelukker andre former for værker.

Endvidere har vi foretaget samme slags optælling af materialer og teknikker, således vi kunne skabe et overblik over de hyppigste anvendte.

Vi har udtrukket alle 'NaN' værdierne for at danne overblik over manglende værdier og udregnet summen for hver kolonne. Således kan vi se, at der forekommer adskillige celler i dataframen, hvor angivelse af tilhørende værdi mangler.

Vi har udregnet hvor mange af værkerne, der står som 'public domain', og fundet frem til at det er 245 ud af 537 værker. Således er der copyright rettigheder på hovedparten af værkerne i SMKs database.

## **Opgave 5**

### **Producering af dataframe over udvalgt data**

Vi har valgt at fokusere på acquisition\_date\_precision, da det viser årstallet for hvornår værket antageligvis blev overleveret til en kulturinstitution og/eller arkiveret

Først har vi lavet en optælling af disse årstal og fremstillet dem i procent. Dernæst har vi lavet en dataframe bestående af årstal i en kolonne og procentvise optællinger i en anden.

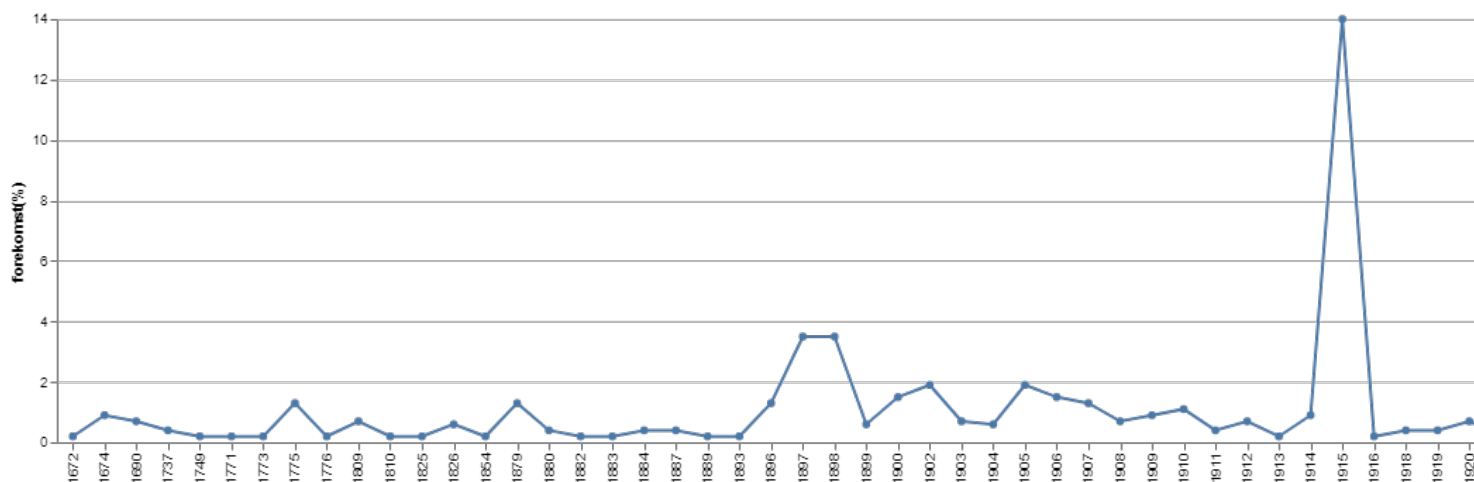
## **Opgave 6**

### **Visualisering af data**

Fra altair biblioteket har vi hentet funktionen, der kan visualisere vores dataframe. Vi har indsat dataframen og angivet år og forekomst som x og y i grafen.

Nedenstående er et udsnit af vores endelige graf. Vi har beskåret denne til fordel for det visuelle, da den fulde graf er mindre overskuelig. Den fulde graf kan findes i bilag 2.

Som fremvist i grafen kan man observere at denne falder og stiger flere steder (Bilag 2). Dog er den største stigning i 1914, der fortsætter indtil 1915, hvorefter den drastisk aftager i 1916. Dette kan bl.a. skyldes, at SMK indtil videre har digitaliseret flest værker fra denne periode og betyder nødvendigvis ikke, at der var flest produktioner af statuetter.



## Litteraturliste

Digitalbevaring.dk (u.å.). Metadata - Hvad er metadata, og hvorfor er de vigtige for digital bevaring?. Lokaliseret d. 2 december 2019 på: <https://digitalbevaring.dk/viden/metadata/>

Dublin Core Metadata Initiative (2019). DCMI Metadata Terms. Lokaliseret d. 29 november på: <https://www.dublincore.org/specifications/dublin-core/dcmi-terms/>

Hack4dk (u.å.). Lokaliseret d. 9. december 2019 på: <https://hack4.dk/>

SMK (u.å.). SMK's API (beta-version). Lokaliseret d. 28 november 2019 på: <https://www.smk.dk/article/smk-api/>

SMK Fridays: Deler kunsten (u.å.). Lokaliseret d. 9. december 2019 på: <https://www.smk.dk/event/smk-fridays-29-november/>

SMK (u.å.). SMK Open. Lokaliseret d. 28 november 2019 på: <https://www.smk.dk/article/smk-open/>

SMK (u.å.). Vizgu. Lokaliseret d. 9 december 2019 på: <https://www.smk.dk/article/vizgu/>

Vizgu (2019). Vizgu. Lokaliseret d. 9 december 2019 på:

<https://vizgu.com/>

## Bilag

Alle bilag er lavet i JupiterLab.

## Bilag 1

Rensning af datasættet

### Før rens af datasættet

id	created	modified	acquisition_date_precision	responsible_department	content_description	frame_notes	materials	object_names	part_of	...	image_iif_id
0	1180035377_object	2019-08-07T05:10:34Z	2019-08-12T09:33:32Z	1898-12-31	Samling og Forskning (KAS)	[Erstattet med KAS2229]	[Bagklædning: false, Mikroklimaramme: false]	[('material': 'gips')]	[('name': 'statuette')]	[KAS2229, ORIG3096]	NaN
1	1180070230_object	2019-08-07T07:24:30Z	2019-08-08T08:29:02Z	1976-12-31	Samling og Forskning (KMS)	[Puys er et fiskerleje i nærheden af Dieppe.]	[Bagklædning: false, Mikroklimaramme: false]	[('material': 'bronzé')]	[('name': 'skulptur')]	NaN	NaN
2	1180032239_object	2019-08-07T05:01:38Z	2019-08-12T09:32:26Z	1915-12-31	Samling og Forskning (KAS)	NaN	[Bagklædning: false, Mikroklimaramme: false]	[('material': 'gips')]	[('name': 'statuette')]	[ORIG3101]	NaN
3	1180081678_object	2019-08-07T08:11:07Z	2019-08-12T09:50:58Z	1915-12-31	Samling og Forskning (KAS)	NaN	[Bagklædning: false, Mikroklimaramme: false]	[('material': 'gips')]	[('name': 'statuette')]	[ORIG3097]	NaN
4	1180014421_object	2019-08-07T04:07:35Z	2019-08-08T08:33:53Z	1925-12-31	Samling og Forskning (KAS)	NaN	[Bagklædning: false, Mikroklimaramme: false]	[('material': 'gips')]	[('name': 'statuette')]	[ORIG3104]	NaN
...	...	...	...	...	...	...	...	...	...	...	...
532	1180058626_object	2019-08-07T06:37:21Z	2019-08-08T07:06:23Z	1968-12-30	Samling og Forskning (KKS)	NaN	[Bagklædning: false, Mikroklimaramme: false]	NaN	[('name': 'tegning')]	NaN	NaN
533	1180017629_object	2019-08-07T04:17:22Z	2019-08-08T08:33:58Z	1970-12-31	Samling og Forskning (KKS)	NaN	[Bagklædning: false, Mikroklimaramme: false]	NaN	[('name': 'tegning')]	NaN	NaN
534	1180060426_object	2019-08-07T06:44:35Z	2019-08-08T11:22:07Z	1968-12-30	Samling og Forskning (KKS)	NaN	[Bagklædning: false, Mikroklimaramme: false]	NaN	[('name': 'tegning')]	NaN	NaN
535	1180004533_object	2019-08-07T04:31:43Z	2019-08-12T09:22:53Z	1690-01-01	Samling og Forskning (KMS)	NaN	[Bagklædning: false, Mikroklimaramme: false]	[('material': 'alabast')]	[('name': 'friskulptur')]	NaN	NaN
536	1180022529_object	2019-08-07T04:31:43Z	2019-08-12T09:29:04Z	1672-12-31	Samling og Forskning (KMS)	NaN	[Bagklædning: true, Mikroklimaramme: false]	[('material': 'lærred'), ('material': 'olie')]	[('name': 'maleri')]	[KMS3076]	https://lip.smk.dk/iif/p2/KMS3076.tif.recons... https://lip.smk.dk/iif/p2/K

537 rows x 49 columns

### Efter rens af datasættet

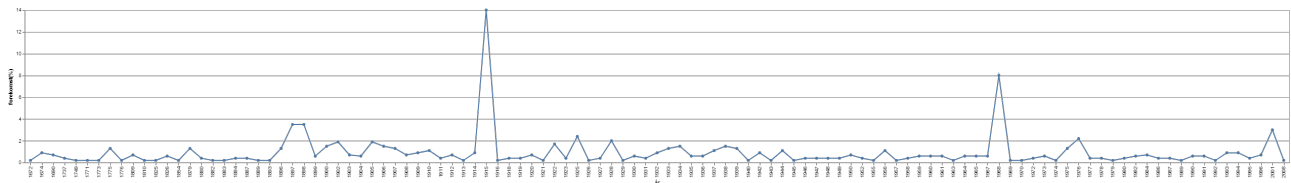
id	created	modified	acquisition_date_precision	responsible_department	content_description	frame_notes	materials	object_names	part_of	...	object_number	public_domain
0	1180035377_object	2019-08-07T05:10:34Z	2019-08-12T09:33:32Z	1898-12-31	Samling og Forskning (KAS)	[Erstattet med KAS2229]	[Bagklædning: false, Mikroklimaramme: false]	[('material': 'gips')]	[('name': 'statuette')]	[KAS2229, ORIG3096]	KAS384	False
1	1180070230_object	2019-08-07T07:24:30Z	2019-08-08T08:29:02Z	1976-12-31	Samling og Forskning (KMS)	[Puys er et fiskerleje i nærheden af Dieppe.]	[Bagklædning: false, Mikroklimaramme: false]	[('material': 'bronzé')]	[('name': 'skulptur')]	NaN	KMSHm3	True
2	1180032239_object	2019-08-07T05:01:38Z	2019-08-12T09:32:26Z	1915-12-31	Samling og Forskning (KAS)	NaN	[Bagklædning: false, Mikroklimaramme: false]	[('material': 'gips')]	[('name': 'statuette')]	[ORIG3101]	KAS1910	False
3	1180081678_object	2019-08-07T08:11:07Z	2019-08-12T09:50:58Z	1915-12-31	Samling og Forskning (KAS)	NaN	[Bagklædning: false, Mikroklimaramme: false]	[('material': 'gips')]	[('name': 'statuette')]	[ORIG3097]	KAS1906	True
4	1180014421_object	2019-08-07T04:07:35Z	2019-08-08T08:33:53Z	1925-12-31	Samling og Forskning (KAS)	NaN	[Bagklædning: false, Mikroklimaramme: false]	[('material': 'gips')]	[('name': 'statuette')]	[ORIG3104]	KAS2049	False

5 rows x 26 columns



## Bilag 2

Visualisering af den udvalgte data med procent i en graf.



## Kode

# Opgave 3: Ved hjælp af Pandas og request, importere nu jeres valgte datasæt.

```
import requests
```

```
import pandas as pd
```

```
import numpy as np
```

```
from pandas.io.json import json_normalize
```

```
api_search_url = 'https://api.smk.dk/api/v1/art/search'
```

```
# Vi opretter en 'dictionary' og angiver værdierne for SMKs API's parametre.
```

```
# Vi har valgt at vores nøgle-søgeord skal være statuetter.
```

```
params = {  
    'keys': 'statuette',  
    'rows': 1000,  
}
```

```
# For at kunne bruge json beautifier, sørger vi for, at det står i JSON-format.
```

```
params['encoding'] = 'json'
```

```
# Her anmodes om vores søgning med SMKs API-URL samt de valgte parametre.
response = requests.get(api_search_url, params=params)

# Således kan vi printe URL'en:
print('Here\'s the formatted url that gets sent to the smk API:\n{}\n'.format(response.url))

json = response.json()

df = json_normalize(json['items'])

# Vi tjekker kolonnerne i head, her kan vi se, at der er 49 kolonner
df.head()

# Her kan vi se alle data-typerne
df.dtypes

# Fjerner kolonner
df.drop(['iiif_manifest', 'object_history_note', 'image_native', 'image_thumbnail', 'has_image', 'related_
objects', 'work_status', 'production_dates_notes', 'credit_line', 'current_location_name', 'content_person'
, 'distinguishing_features', 'alternative_images', 'image_mime_type', 'image_iiif_id', 'image_iiif_info',
'image_width', 'image_height', 'image_size', 'image_cropped', 'image_orientation', 'exhibitions',
'labels'], inplace=True, axis=1)

# Navngiver kolonner
df.rename(columns={'created': 'reg_year'}, inplace=True)
df.rename(columns={'object_names': 'Type'}, inplace=True)
df.rename(columns={'number_of_parts': 'number'}, inplace=True)
df.rename(columns={'acquisition_date_precision': 'acqdate'}, inplace=True)

# Her kan vi se, at der nu er 26 kolonner
print(df.head())

# Formen på dataframen: 537 rækker og 26 kolonner
```

```
print(df.shape)
```

```
# Opgave 4: Udtræk og beregn
```

```
# Finder NaN værdier/missing values og returnerer summen.
```

```
# Således kan vi se, at der forekommer celler med manglende værdier
```

```
print(df.isnull().sum())
```

```
# Antal af værker, der er registreret som public domain.
```

```
df['public_domain'].sum()
```

```
# Kan se alle årstallene
```

```
alleårstal = []
```

```
for i in range(len(df['acqdate'])):
```

```
    alleårstal.append(df['acqdate'][i][0:4])
```

```
    alleårstal.sort()
```

```
from collections import Counter
```

```
Counter(alleårstal)
```

```
# Kun unikke årstal i datasættet
```

```
unikkeårstal = []
```

```
def unique(alleårstal):
```

```
    x = np.array(alleårstal)
```

```
    unikkeårstal.extend(np.unique(x))
```

```
unique(alleårstal)
```

```
unikkeårstal #111
```

```
# Fordelingen af forskellige typer af værker inden for nøgleordet, statuettes.
```

```
df['Type'].value_counts()
```

```
# Fordelingen af værker registreret som public domæne. False 292 || True 245. Fleste er ikke.
```

```
df['public_domain'].value_counts()
```

```
# Hvor mange er public i alt
```

```
df['public_domain'].sum()
```

```
# Optællinger
```

```
df['acqdate'].str.extract(r'^(\d{4})', expand=False).value_counts() #vi udtrækker de specifikke årstal
```

```
og foretager en optælling
```

```
df['materials'].value_counts()
```

```
df['techniques'].value_counts(normalize=True)
```

```
# Opgave 5: Ny dataframe med value_counts
```

```
# Vi har lavet en procentvis optælling af acquisition date
```

```
optæl_årstal = df['acqdate'].str.extract(r'^(\d{4})',
```

```
expand=False).value_counts(normalize=True).mul(100).round(1) #får kun optællingerne
```

```
nydfdate = pd.DataFrame(optæl_årstal)
```

```
# Her har vi genereret en dataframe med årstal og optællingen
```

```
opdeltdf = pd.DataFrame({'år': nydfdate.index, 'forekomst(%)': optæl_årstal})
```

```
# Opgave 6: Visualisering af den nye dataframe. Dette fremvises i Jupyter Lab.
```

```
# Altair biblioteket importeres.
```

```
import altair as alt
```

```
# Ved x har vi sat år, som er den horisontale linje
```

```
# Ved y har vi sat forekomst i procent, dvs. optællingen, som vi indsatte i en dataframe, og vises i  
den vertikale linje
```

```
alt.Chart(opdeltdf).mark_line(point=True).encode(  
    x='år',  
    y='forekomst(%)',  
    tooltip= [alt.Tooltip('år'), alt.Tooltip('forekomst(%)')] #for hover effect  
) .properties(width=2500, height=300)
```

## Etiske overvejelser

### Portfolio 1

I denne opgave har vi arbejdet med deskriptiv statistik, hvori vi har visualiseret fordelingen af passagerer i forhold til klasse og omkomne/overlevende. Dette er særdeles praktisk, hvis man ønsker at skabe overblik over større datamængder. Videre kan det benyttes inden for adskillige videnskaber såsom sociologi med forskningsprojekter, der også trækker på kvantitative teknikker. Når større datamaterialer er blevet tilgængeliggjort til offentlig anvendelse, bliver den enkelte forsker dog stillet over for en række etiske overvejelser og principper som kommer til at udgøre en væsentlig del af den etiske beslutningsproces. De forskellige princippers relevans vil variere i forhold til den problemstilling, som datamaterialet medbringer (Code of Ethics and Professional Conduct, 2018). I relation til datasættet indeholder det informationer om mennesker, der var passagerer på skibet under den tragiske hændelse i 1912. I denne cvs-fil kan man generere en tabel over antallet af omkomne og overlevende og dernæst sætte det i forhold til deres klasse eller billetpris. Således præsenterer dette datasæt en problemstilling omkring ansvar. Det blev fastlagt af 'European Science Foundation', at alle mennesker ligeledes skal behandles med respekt inden for data science (The European Science Foundation, 2001). På det grundlag kan man diskutere, hvorvidt dette princip bliver efterlevet, idet man har kvantificeret adskillige mennesker, der mistede livet i denne hændelse.

Personoplysninger betegnes som informationer, der kan henføres til en bestemt person. Herunder kan disse oplysninger bl.a. opdeles i almindelige og følsomme som behandler henholdsvis navne og genetiske data (Hvad er personoplysninger, n.d.).

I dette tilfælde har man offentliggjort passagerens navne, aldre og klasse. Dette kunne være et eksempel på personhenførbare data, eftersom det indbefatter identifikationsoplysninger samt oplysninger omkring økonomiske forhold i forbindelse med priserne på deres billetter. Der er ligeledes tale om følsomme persondata, eftersom datasættet indeholder oplysninger om deres

køn. For at undgå overtrædelse af retningslinjer mhp. persondata, kan forskeren eksempelvis sikre at anonymisere de personer, der udgør empirien.

## Portfolio 2

Denne opgave indbefattede bl.a. tokenisering samt analyse og visualisering af ord mhp. hyppigheden af deres forekomster. Således kunne vi observere, hvilke emner der var mest debatterede i forhold til en bestemt person. Green og Cross (2017) har adopteret denne metode som supplerende værktøj til at afdække politiske agendaer i taler (s. 78). Derved kan forskellige videnskabsområder drage fordel af at anvende disse metoder til eksempelvis at undersøge sociale fænomener.

Det er dog ikke kun persondata, som forskeren skal være opmærksom på i databehandlingen. Vi arbejdede nemlig med the guardians nyhedsartikler, hvori vi valgte at visualisere de mest omtalte emner i forbindelse med 'Boris Johnson'. Dette kan være problematisk, idet man ikke udelukkende kan drage konklusioner på baggrund af data. Vi kunne anskue hvilket temaer, der var mest fremtrædende i vores datakollektion. Dette krævede dog menneskelig indblanding i form af fortolkning. Vi vidste nemlig, at størrelsen på vores dataframe sandsynligvis ville generere en meget begrænset 'topic model'. Dette betød, at det nødvendigvis ikke var hvert eneste ord fremvist i vores 'word cloud', der bar den samme betydning. Desuden ville vores måling være præget af vores egen forudindtagethed, da vi havde en specifik ide om, hvad vi kunne forvente at observere i vores resultater.

I henhold til ACM's kan et eksperiment ikke være konkluderende, hvis det ikke kan reproducere med de samme resultater (Artifact Review and Badging, 2018). Hvis vores undersøgelse blev foretaget af et andet team og med et andet setup, eksempelvis en længere periode end 2 måneder, ville det sandsynligvis have produceret andre resultater, da vores 'topic model' var et meget lille udtræk af virkeligheden.

Ifølge Hampton (2015) omfatter open science transparens og åben tilgængelighed (Hampton, 2015, s. 1-2). Med henblik på the Guardians åbne API, der har tilgængeliggjort store mængder af nyhedsartikler, har de således efterlevet dette princip omkring åbenhed. Dog kan det være risikabelt at måle og vurdere fænomener og personer i samfundet på baggrund af data fra nyhedsartikler. Således er man nødsaget til at være varsom med denne form for data og sikre, at ens praksis og dømmekraft omfatter respekt for det enkelte menneske, som artiklerne omtaler.

### Portfolio 3

Kulturarvsdata omfatter bl.a værker, aviser, metadata, som forskeren kan anvende til statistiske beregninger og visualisering af data i projekter. Eksempelvis kan historikere undersøge tendenser og mønstre, der kulminerer i bestemte perioder samt de bagvedliggende årsager. Allen og Sieczkiewicz (2010) beskriver ligeledes nyhedsartikler som værende væsentlige ressourcer for historikere i deres datakollektion (s. 1).

I relation til portfolio 3 har vi arbejdet med data fra SMKs API. Vores udvalgte datamængde frembragte problemstillinger omkring ophavsrettigheder, da vi kunne observere adskillige metadata, der angav, hvorvidt værket var anført som 'public domain'. Videre kunne man undersøge, om de var underlagt 'creative commons' licensen eller copyright. Dette er en væsentlig overvejelse, som forskeren må varetage, nemlig, at respektere ophavsrettigheder, patenter et cetera (Code of Ethics and Professional Conduct, 2018). Dog fremstod hovedparten som materialer, der er til offentligt domæne. Dette indbefatter, at de er ikke er underlagt ophavsrettigheder og kan anvendes frit (Code of Ethics and Professional Conduct, 2018). Dermed kan SMKs åbne og digitaliserede datamateriale medvirke til vidensopdagelse såvel som innovation. Dette benævnes af Wilkinson (2016) som et resultat af god datastyring med henblik på FAIR principperne; tilgængelighed, interoperabilitet, genanvendelighed og identificering (s. 1). Materialerne var tilgængelige for alle vha. API'en og videre anvendte SMK standard termer i forhold til at identificere de forskellige metadata, som vi redegjorte for i opgaven. Det var ligeledes muligt at skelne mellem de forskellige værker, idet de var tildelt unikke ID.

### Referencer:

Artifact Review and Badging. (2018). Retrieved 1 December, 2019, from:

<https://www.acm.org/publications/policies/artifact-review-badging>

Code of Ethics and Professional Conduct. (2018). Retrieved 1 December, 2019, from:

<https://www.acm.org/code-of-ethics>

Allen, R., & Sieczkiewicz, R. (2010). How historians use historical newspapers. *Proceedings of the American Society for Information Science and Technology*, 47(1), pp. 1–4.

<https://doi.org/10.1002/meet.14504701131>

The European Science Foundation. (2001). The European Code of Conduct for Research Integrity. Strasbourg: Ireg.

Hvad er personoplysninger. (n.d.) Retrieved 1 December, 2019, from:

<https://www.datatilsynet.dk/generelt-om-databeskyttelse/hvad-er-personoplysninger/>

Greene, D. & Cross, J.P. (2017). Exploring the political agenda of the european parliament using a dynamic topic modeling approach. *Political Analysis*, 25(01), pp. 77–94.

Hampton, S., Anderson, S., Bagby, S., Gries, C., Han, X., Hart, E., ... Zimmerman, N. (2015). The Tao of open science for ecology. *Ecosphere*, 6(7), pp. 1–13. <https://doi.org/10.1890/ES14-00402.1>

Wilkinson, M. D., Dumontier, M., Aalbersberg, IJ. J., Appleton, G., Axton, M., Baak, A., ... Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3, [160018]. <https://doi.org/10.1038/sdata.2016.18>