Frontcover to Hand-ins from ....

by Jeannette Ekstrøm

studentID - dgk432

email - jeek [at] dtu.dk

Here you find "concatenated" all the "hand-ins" from class (BATV) 4900-E19 as one pdf
(INF Open Data Science (BATV) at Copenhagen University.)

See more at page 7 about the class
https://hum.ku.dk/uddannelser/aktuelle_studieordninger/informationsstudier/informationsvidenskab_og_kulturformidling_batv

Portfolio 1 (from page 2-9)

Portfolio 2 (from page 10 - 24)

Portfolio 3 (from 25 - 47)

including output

- all hand-ins are available at https://github.com/BA-ODS2019/dgk432
- please run the scripts from there..

Portfolio opgave nr. 1

Det datasæt vi har fået foræret til at løse Portfolio nr. 1 handler om Titanic færgens forlis (https://github.com/BA-ODS2019/dgk432/blob/master/data/titanic.csv). Opgaven går ud på at foretage nogle enkle analyser på indhold i datasættet ved brug af Python og Pandas biblioteket, bl.a.

Opgaven lyder som følgende:

1. Åben filen i en tekst-editor og se på indholdet.

   Hvilke data typer kan i identificere? (tekst / tal / typer af tal mv)

   Mangler der data?

2. Ved at bruge panda – api skal i importere data til en dataframe.

   Beskriv nu data-sættet med de funktioner der findes i Pandas til beskrivelse af en dataframe (se f.eks i https://datacarpentry.org/python-socialsci/08-Pandas/index.html)

3. Og nu er det op til jer at udtrække og beregne på data i filen som kan give os informationer om de personer der var involveret i ulykken fx hvor mange overlevede?, gennemsnitsalder og medianen alder på personerne? (Dvs deskriptiv statistik)

4. Findes der personer med samme efternavn?

5. I skal nu lave en pivot-tabel som viser hvor mange der rejste på hhv 1., 2. og 3. klasse.

   Hvilken rejseklasse havde flest omkomne?

Løsningen på ovenstående opgave kan naturligvis ses i tilhørende python script - som kan findes via https://github.com/BA-ODS2019/dgk432/blob/master/P1/portfolio_1.py

Ad spm 1

Indledningsvis har jeg blot åbnet filen via min texteditor (Visual Code Studio i mit tilfælde) og med "menneskelige intellekt og egne øjne" tjekket datasættet for indhold, herunder kolonne overskrifter mv. Datasættet består at både tekst og heltal/decimaltal, og der synes ikke at mangle nogle data i de mange rækker af data.

Ad spm. 2

Her skal vi ved hjælp af data analyse processer skabe os et (digitalt) overblik over indholdet og formatet af den delte *.csv fil. Denne sidste process foregå ved hjælp af Python, samt to relevante "add on biblioteker", kaldet Pandas og Numpy. Pandas er et Open Source Python bibliotek ideelt til indledende data analyse. Det gør det muligt at arbejde i Python med "spreadsheet like data for fast data loading, manipulating, aligning and merging" (https://learning.oreilly.com/library/view/pandas-for-everyone/9780134547046/). Pandas tilføjer nye datatyper til selve Python programmet, dels *Series* og *DataFrame*. DataFrame er unikt for Pandas, og repræsenterer hele ens spreadsheet/data i en RAMME, mens Series blot fokuserer på een enkelt kolonne af denne Dataframe. Numpy er ligeledes et Open Source bibliotek til Python, her er fokus på store, multi-dimensionelle "arrays og matrices" (tal), tilsat en pæn, stor samling af matematiske funktioner. Pandas og Numpy er afhængige af hinanden og begge

biblioteker skal/bør importeres til ens Python installation for optimal udnyttelse. Datasættet er i første omgang undersøgt via en Data Carpentry lesson https://datacarpentry.org/python-socialsci/08-Pandas/index.html om basis Data Analyse vha Pandas.

Ad spm. 3 - 4 og 5, så fremgår resultaterne af scriptet

Men eftersom at data automatisk indlæses som en DataFrame, så kan man let via enkle statements kan danne sig et overblik over størrelsen på ens datasæt eks. >>> print(titanic.shape) giver resultatet 8 kolonner/887 rækker, og >>> print(titanic.columns) giver mig overskrifterne på de 8 kolonner, som jeg efterfølgende skal arbejde med.

Arbejdet med data (store og små datasæt)

Arbejdet med store datamængder (eller små) handler om mere en blot ren kodning. Data Analyse og Data "cleaning" aspektet har vist sig at tage en stor del af den tid, man afsætter til at arbejde med data. Det er vigtig at forstå datasættet uanset størrelse. For at kunne foretage reelle "data manipulationer" kræver det at man har indsigt, overblik og kender sine data grundigt, da forkert kodning/programmering kan få fæle konsekvenser og skæve resultater, hvis ikke man bearbejder data korrekt. Derfor er god forståelse for kode samt god forståelse og indsigt i ens data vigtige parametre inden data analysen foretages. Desuden skal man behandle data ordentligt, både ud fra data etiske overvejelser som anonymisering hvor det kan være krævet (interviews, surveys), men også korrekt citering/kreditering af data er vigtig, når man evt. finde åbne datasæt via Scraping eller API løsninger, men også når man finder sine datasæt fra repositories som Zenodo, Figshare eller Dryad for selv at arbejde videre med disse tilgængelige data. I tilfældet med Portfolio 1 ***Titanic datasættet*** så kan man finde dette og lignende datasæt fra Titanic mange steder via nettet - dels har sitet Kaggle link til datasættet (https://www.kaggle.com/c/titanic-gettingStarted/data) men også et slutbrugerprodukt som databasic.io anvender titanic data (https://www.databasic.io/en/wtfcsv). Vi har ikke fået en konkret kilde anvist til hvor vores datasæt er hentet fra, men det er tydeligt, at portfolio 1 ***Titanic datasættet*** ikke er helt det samme som ovenfor nævnte. Vores sæt har færre kolonner og rækker.

I Open Science Training Handbook, kapitel 4 (https://open-science-training-handbook.gitbook.io/book/open-science-basics/reproducible-research-and-data-analysis) beskrives de 5 trin af den videnskabelige metode:

- 1. Formulating a hypothesis
- 2. Designing the study
- 3. Running the study and collecting the data
- 4. Analyzing the data
- 5. Reporting the study

Open Science har fokus på åbenhed, og hver af de 5 trin skal beskrives klart vha åben dokumentation, så studiet/forskningen gøres genneskuelig og transparent. I første omgang for at imødekomme "bedrageri og videnskabelig uredelighed", men også for at kunne genbruge nyttige datasæt til ny forskning og studie. I gamle dage skulle metodeafsnittet i et paper beskrive, hvordan data var indsamlet og anvendt, fremadrettet skal data også beskrives udførligt (metadata-beskrivelser til samtlige dele af forsknings-data-sæt). Metadata beskrivelserne omfatter parametre som ophav, ansvarlig, brugslicens, redistribuering, lagring, reproduktion - hvilket i dag betyder, at man kan (og skal) beskrive sine datasæt, uanset om selve datasættet i sig selv er frit tilgængelig eller ej. En måde man kan gøre dette er ved anvendelse af "FAIR principperne". FAIR står for FINDABLE, ACCESSIBLE, INTEROPERABLE, og REUSEABLE (https://en.wikipedia.org/wiki/FAIR_data) - dvs.

data skal være beskrevet ud fra nogle standarder som gør det let at gennemskue hvad det er for data, hvordan man evt. kan få adgang, hvordan det vedligeholdes osv.

Overvejelser her på falderebet

Kurset er jo ikke et decideret programmerings-kursus, og læringskurven var stejl. Færdigheder i basal programmeringsforståelse viste sig at være meget nødvendig for at komme i mål med opgaverne. Men i løbet af kursusforløbet har de mange øvelser vist sig nyttige. Respekten for at Data Analyse tager tid er helt klart noget som jeg tager med mig fra kurset. Og det at kunne planlægge og udføre analyseopgaver kan først for alvor omfavnes, når man har de teoretiske og håndværksmæssige begreber på plads. Dataforståelse, kode-flair og viden om hvad man må med data og hvordan spiller en lige så stor rolle - i min verden (universitetsbibliotek) kalder vi den samlede pakke for DATA LITERACY. Der er "indirekte" krav til Data Literacy på alle niveauer, og derfra hvor min arbejdsverden udspringer, så mærkes det tydeligt, at udviklingen hen i mod større data forståelse, ønske om support og servicetiltag i forbindelse med Data Management, Data håndtering (licenser, etik, citationer, storage, curaation) bliver stadig større.

Det at nødvendigt at forstå flere parametre end blot adgang til data og interesse for programmering. At nå til det stadie, hvor man kan gennemskue kode, kan "formulere de rette spørgsmål til "fejlfinding, support, råb om hjælp" via eks. https://stackoverflow.com/ og nettet generelt, så er man nået langt. Men derfra kræves cases og masser af øvelse. For at nå næste læringsniveau - at sætte teori og læring ind i den praktiske arbejdsopgave - er man nødt til at have konkrete opgaver og bare blive ved med at anvende det lærte.

Enkle scripts og programmerings-snippets udarbejdet i eksempelvis et program som Python kan lette denne data-analyse proces, gøre den mere automatisk og re-producerbar. Åbenhed om den data-analyse-metode man har anvendt medfører, at andre vil kunne genskabe lignende, måske endda nøjagtig samme resultater - til gavn for samfundet, til nytte for videnskab & studier.

Kurset har gjort, at jeg nu føler mig bedre rustet til de mange nye opgaver, som kan løses med enkle scripts og lidt programmerings-krydderi. Det at kunne læse "et styk kode uden af flippe ud" og samtidig kunne omsætte dette til noget praktisk anvendeligt har været målet. Mine primære arbejdsopgaver handler om og har fokus på læring, formidling og videndeling. At være DATA SAVVY på et teknisk universitet er nødvendig. Og det er min overbevisning, at lidt programmeringskendskab kan lette mange opgaver i dagens forskningsbibliotek. Både overfor brugerne, men også i de mange daglige driftsopgaver, som finder sted i nutidens fag,- forsknings,- og universitetsbiblioteker. At kunne tænke smarte løsninger ud fra "trivielle og gentagne opgaver" vil helt klart være mit fokus fremadrettet.

portfolio_1.py

- from https://github.com/BA-ODS2019/dgk432/blob/master/P1/portfolio_1.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 23 21:10:53 2019

@author: je
"""

# Åben og læs filen ...
import pandas as pd
import numpy as numpy
import matplotlib.pyplot as plt
import seaborn as sns
import re as re

# Besvarelser angående spm. 1-3 fra Portfolio 1 opdraget

# Min variable sættes til *titanic* - som jeg så læser ind som en Pandas
Dataframe.
# Jeg har valgt at indsætte ektra "print statements(strings) til bedre
overblik/forståelse.

print('\n\Indlæsningen af min csv fil')
# her indlæses min fil fra mit drev, data kan hentes
# via https://github.com/BA-ODS2019/dgk432/tree/master/data
titanic = pd.read_csv("../data/titanic.csv")

# Første step er at undersøge, hvilken type (datasæt) jeg har med at gøre

print('\n\Dokumentation for, at mit datasæt rent faktisk er en Pandas
DataFrame ' )
print(type(titanic))

# Derefter begynder jeg min analyse af datasættet, dels for at undersøge,
hvilke typer
# data jeg har med at gøre, men også for at lære data at kende, så jeg kan
# uddrage enkelte visualiseringer til sidst i processen

# Kilder der er anvendt er:
# https://www.shanelynn.ie/using-pandas-dataframe-creating-editing-
viewing-data-in-python/
# og https://pandas.pydata.org/pandas-docs/stable/index.html

print('\n\Overskrifter på kolonnerne: ' )
# Navnene/værdierne på de overskrifter, som findes i datasættet
# Disse værdier kan der beregnes og analyseres på senere hen
print(titanic.columns.values)
```

```
print('\n\Total antal rækker og kolonner i min Pandas DataFrame: ')
# Hvor mange rækker og kolonner beregnes med shape funktionen
print( '\n\Antal rækker og kolonner: ')
print(titanic.shape)
print('\n\Dimensionerne i min DataFrame: ')
print(titanic.ndim)

print('\n\Samlet antal rækker og total antal kolonner:  ')
# Samlet antal rækker/entries?
print(len(titanic))
print('\n\Samlet antal kolonner: ')
# Antal kolonner?
print(len(titanic.columns))

print('\n\Total antal celler: ')
#Hvor mange 'cells' i tabellen
print(titanic.size)

# Kolonnernes data type - (int, float, string, object...)
print( '\n\Datatypen fra kolonnerne (om det er integer, float eller
object/string) : ')
print( 'kort overblik via funktionen print(titanic.dtypes) :')
print(titanic.dtypes)

# .. med nedenstående kode snip (titanic.info) får man samme overblik som
alle
# de ovenstående koder, inkl type af data og overblik til videre
bearbejdning
print('\n\Hvilken type data min DataFrame indeholder: ')
print( 'mere dybdegående indsigt via funktionen >>>print(titanic.info) :')
print(titanic.info())

# Det er tydeligt, at funktionen >>>print(titanic.info) giver flere
informationer end
# >>>print(titanic.dtypes)

print('\n\nSidste for-analyse funktioner :')
# Beregninger til at undersøge data nærmere, inkl samlet antal, hele
rækker fra DataFrame
print(titanic.head()) # viser kun de 5 første hits/rækker i DataFrame
print(titanic.tail()) # viser de 5 sidste rækker i DataFrame
print(titanic.describe()) # større samlet overblik inkl kolonne data

# Break inden beregning, hvor der anvendes "Descriptive statistics"
print( '\n\nAntal overlevende: ')
# for at finde antal overlevende skal man navigere ned til den rette
kolonne
# og få output derfra - 'Survived' er enten 0 eller 1. 1 = overlevet.
print(titanic['Survived'].sum()) # Total sum of the column values -
overlevende

print( '\nGennemsnitsalder: ')
# sammen princip som ovenstående - nu blot med kolonnen AGE
print(titanic['Age'].mean()) # Mean of the column values - alder..
```

```
print( '\n\nMedian - alder: ')
print(titanic['Age'].median()) # Median of the column values - medianen..

# Find antal kvinder ...
females = titanic[titanic['Sex'] == 'female']
print('\n\nAntal kvinder: ')
print(females)

# Find antal mænd ...
males = titanic[titanic['Sex'] == 'male']
print('\n\nAntal mænd: ')
print(males)

print('\n\nHvordan er fordelingen mellem passagerklasse og overlevende:' )
# For at få fordelingen på passagere på rejseklasse skal man gruppere på..
og sorte værdien
# passagerer fra Pclass and Survived kolonnerne ..
print (titanic[['Pclass', 'Survived']].groupby(['Pclass'],
as_index=False).mean().sort_values(by='Survived', ascending=False))

print('\n\nHvordan er fordelingen så mellem køn og overlevende? : ')
# En lille ændring i forhold til ovenstående
# nu overlevende fra Sex og Survived kolonnerne -
print (titanic[['Sex', 'Survived']].groupby(['Sex'],
as_index=False).mean().sort_values(by='Survived', ascending=False))

# Antal personer på klasse 1, 2 og 3
print('\n\nantal personer på henholdsvis klasse 1, 2 og 3')
print(titanic.groupby('Pclass').count())

# Spørgsmål 4 i opdraget handler om, hvorvidt der er personer med samme
efternavn
# For at kunne foretage denne beregning, skal man anvende split funktionen
# dvs. først split på teksten(string), dernæst beregnes variablen i
forhold til value_count
print( '\n\nSortering pr efternavn: ')
print( '\n\nVed at tage sidste værdi/entry fra kolonnen Name har jeg fået
en liste over efternavne' )

last_names = titanic['Name'].str.split().str[-1]
print(last_names)

print('\n\nHvor mange rejsende har samme efternavn')
print(last_names.value_counts())

# Spørgsmål 5 i Portfolio 1 opgaven lød på at lave en Pivot tabel,
# som viser, hvor mange der rejste på henholdsvis 1, 2 og 3 klasse

print('\n\nAntal overlevende pr. klasse vises som Pivot tabel: ')
pt1 = titanic.pivot_table(columns = 'Pclass', values = 'Survived',
aggfunc='sum')
print(pt1)
pt1.plot(kind='bar')
```

```
plt.title("antal overlevende pr. klasse")
plt.savefig("SurvivorsPrClass.png")


print('\n\nAntal overlevende pr. køn vises som Pivot tabel: ' )
pt2 = titanic.pivot_table(columns = 'Sex', values = 'Survived',
aggfunc='sum')
print(pt2)
pt2.plot(kind='bar')
plt.title("antal overlevende pr. køn")
plt.savefig("SurvivorsPrSex.png")


# Ved også at anvende visualisering via Seaborn biblioteket kan man
tydeligt se, at der
# var mange flere mænd ombord end kvinder - inspiration snuppet fra Kaggle
# https://www.kaggle.com/startupsci/titanic-data-science-solutions
print('\n\nVed hjælp af Seaborn visualisering kan man se forskel i antal
mænd og kvinder om bord Titanic - fordelt på alder' )
t = sns.FacetGrid(titanic, col='Sex')
t.map(plt.hist, 'Age', bins=20)


print('\n\nSidste øvelse, BONUS i forhold til oplæg, er at få samlet de
mange forskellige titler, som passagererne har' )

# som lidt ekstra, kan man samle titler, denne inspiration er fundet via
netsøgning
# og ved brug af RegEx..

# En enkelt definition inkl. brug af RegEx
# kilde: https://regex101.com/ blandt andet..
# og https://www.kaggle.com/startupsci/titanic-data-science-solutions til
inspiration

def get_title(name):
        title_search = re.search('([A-Za-z]+)\.', name)
        # If the title exists, extract and return it.
        if title_search:
                return title_search.group(1)
        return ""

print('\n\nTitler, som de forskellige passagerer benytter sig af')
titanic['Title'] = titanic['Name'].apply(get_title)
print(pd.crosstab(titanic['Title'], titanic['Sex']))

print('\n\nSøg og erstat de mange titler så det forenkles: ' )
titanic['Title'] = titanic['Title'].replace(['Lady', 'Countess','Capt',
'Col',\
        'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Other')
titanic['Title'] = titanic['Title'].replace('Mlle', 'Miss')
titanic['Title'] = titanic['Title'].replace('Ms','Miss')
titanic['Title'] = titanic['Title'].replace('Mme','Mrs')
titanic['Title'] = titanic['Title'].replace('Master', 'Mr')
```

```
print('\nPrint til sidst de samlede titler kombineret med hvorvidt de
overlevede:')
print (titanic[['Title', 'Survived']].groupby(['Title'],
as_index=False).mean())
```

Portfolio 2 documentation

- By Jeannette Ekstrøm,
- jeek [at] dtu.dk
- student ID : dgk432

In order to solve the assignment I had to struggle a bit with code.

I had 3 steps I wanted to solve:

1. Getting the data
2. Pre-process the data and answer the questions 2. incl. built vector models, topic models and queries
3. Visualize by use of WordCloud

I decided to split up the process, and having one script for ***Getting the data*** https://github.com/BA-ODS2019/dgk432/blob/master/P2/gettingthedata.py

and a second script ***pre-processing the results and visualizations*** https://github.com/BA-ODS2019/dgk432/blob/master/P2/preprocessing.py

I created 7 functions inspired by the lecture slides and supplementary links & literature we have been through during the themes from Module 2 (Basic Text Analysis, Vector Space Models, Text Classification and Topic Modelling). I used Visual Code Studio and also Spyder as editors, not Jupyter Notebook as the teacher did.

That way I kept a better structure - and a better understand of which ***variables*** I was processing on, as well as an understanding of how I got access to my data.

I used "literature and google" to learn about - how to add create functions (***def***) . ... programming is supposed to reduce workloads and create a nice overview, so instead of performing the same line of code on several different words for counting and analysis, creating a function (incl a few arguments I wanted to return better structured code. Functions are defined and inspired using ***def*** function like this one below:

---

def count_in_list(item_to_count, list_to_search): # define function and give it a name + two arguments

```
number_of_hits = 0                          # define our variable +
assign value zero
for item in list_to_search:                 # we loop over all words in
our list
    if item == item_to_count:               # if we find words equal to
        number_of_hits += 1                 # we add 1 to our variable
[list]
return number_of_hits                       # finally return the result
```

print(count_in_list(' keyword of our choice ', words))

from source: (https://nbviewer.jupyter.org/github/fbkarsdorp/python-course/blob/master/Chapter%202%20-%20First%20steps.ipynb)

Because my data fetch was pretty large I had to create a few "sub-sets" - otherwise the process would take to long. The PATH I did not use was then just hash'ed out, which made it easier for me to process the statements - without much delay The final script is with the complete dataset.

Pre-processing unstructured data like news articles (text mining)

Preprocessing data/text is all about making better sense of the text, you are working with, and is a very common and useful task in text analysis. It's about discovering patterns and examining context, uniqueness and outliers of large chuncks of text, that being newsgroup data, novels like **Jane Austens** *Emma* or like in our case, news articles and feature articles from a well known newspaper like The Guardian.

Tokenization is the process of splitting a text into individual words or perhaps sequences of words (n-grams). (Ignatow & Mihalcea, 2018, page 101-103). Big decisions need to be thought through when using tokenization, one reason being language differences. Splitting words in the english language versus the danish language may see difficult in one iteration. But in most cases - I believe - the data that you want to use will be fetched from one source and therefore in one single language. At least in our Portfolio 2 case I grab news from The Guardian, all in english. In order to look for word patterns and word frequencies you will experience lots of "similar words just spelled a bit different, being plural versions or inflected forms of a word". This special procedure is called ***stemming***, and is used in many search engines behind the scenes, resulting in a better search experience. In this case, I have left out these options (stemming, lemmatization) in my assignment delivery.

One of the first pre-processing was to split all the sentence into a list of words (represented by strings). By issuing the function split Python splits the sentence on spaces and returns a list of words. And returns a list. It makes it possible to access all the individual components using indexes and we can use slice indexes to access parts of the list. Using the split function is a ***quick and dirty*** way to get an overview of a corpus. And the function do unfortunately not take lower and uppercase into consideration. It simply splits sentences into ***strings/words*** by use of ***spaces***, and then return a list of strings/words to work with further. But all this is taken care of by using the Scikit learn library.

Document term matrix - Vector - Topic modelling - Text analysis for text mining..

Treating texts as a list of word frequencies (a vector) also makes available a vast range of mathematical tools developed for studying and manipulating vectors. Something that really is not neccessary to "know in dept". (https://liferay.de.dariah.eu/tatom/working_with_text.html) But the process turn the texts into unordered lists / socalled ***Bag of Words*** (BOW) - that you can start working on.

In text mining we often access collections of documents (open data or otherwise), like in this case with the Guardian news articles. when we "scrape the fulltext" of news articles by use of an API key, we have acquired. The data is OPEN, we need to credit where we got it from, and also accept what to use it for, but the result is, that we receive lots of unstructured data, that we can start to process in order to discover "head and tail" in the context for either study or research.

In our course Python programmering was a large part of the learning objectives, and we had to learn quite a few special Python programming libraries. From Pandas DataFrames, to Numpy mathematics and to more specialised libraries that would assist us in the text classification process.

```
"Latent Dirichlet allocation (LDA) is a particularly popular method for
fitting a topic model. It treats each document as a mixture of topics, and
each topic as a mixture of words. This allows documents to “overlap” each
other in terms of content, rather than being separated into discrete
groups, in a way that mirrors typical use of natural language."
(https://www.tidytextmining.com/topicmodeling.html)
```

LDA was really useful for getting hold of topics in my collection of "top 10 documents" - and it is definitely something I will explore further, after the class. We have "learned" it by example, by googling and by reading literature, next step is to get it to use in practice.

Another method was looking at similaries and frequencies, by use of Tf-idf (term frequency - inverse document frequency) that I before new from the major search-engines. Tf-idf is a way to add algorithms behind search results, in order to receive relevant results - with similar topics/themes. It has been intesting to understand some of what goes on behind the nice search interfaces. In this assigment I used it as thought in class, by looking into patterns and similarities in my search result/downloaded dataset. One think though is, that the algorithm is transparent and the results are reproducible in other contexts - open science rules ....

```
"Corpus exploration = meaning analysing a corpus of text
a pre-processing step for text-mining measures and models = cleaning,
pattern recognition..

Tf-idf is especially appropriate if you are looking for a way to get a
bird’s eye view of your corpus early in the exploratory phase of your
research because the algorithm is transparent and the results are
reproducible - The math behind tf-idf is lucid enough to depict in a
spreadsheet."
(https://programminghistorian.org/en/lessons/analyzing-documents-with-
tfidf#fn:1)
```

I have used # in my script, to explain how and what process I have done.

Inspiration and documentation:

- https://liferay.de.dariah.eu/tatom/preprocessing.html#index-0
- https://www.tidytextmining.com/topicmodeling.html
- https://learning.oreilly.com/library/view/mastering-machine-learning/9781788299879/73015649-db61-4956-a5dd-5116ca59e839.xhtml
- Ignatow & Mihalcea, 2018 https://us.sagepub.com/en-us/nam/an-introduction-to-text-mining/book249451

gettingthedata.py

- from https://github.com/BA-ODS2019/dgk432/blob/master/P2/gettingthedata.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 28 12:36:57 2019

This program only download data from The Guardian
processing the acquired data happens in another file..

@author: je
"""

import os
import json
import requests
from os import makedirs
from os.path import join, exists
from datetime import date, timedelta

# ----------------------
# CONSTANT DEFINITIONS
# --------------------------
# ARTICLE DATA DIRECTORY:
DATADIR = str( os.getenv('OPENDATASCIENCE_DATADIR') )
          # This creates two subdirectories called "theguardian" and
"collection"
ARTICLES_DIR = join( DATADIR,'theguardian', 'collection')

# API Key is defined in the environement.
# Define using (on unix) : export GUARDIAN_OPEN_API_KEY='xxxxx'
MY_API_KEY = os.getenv('GUARDIAN_OPEN_API_KEY')

# Where to get the data from...
API_ENDPOINT = 'http://content.guardianapis.com/search'

# TIME PERIOD TO DOWNLOAD: Update these dates to suit your own needs
START_DATE = date(2018, 1, 1)
END_DATE = date(2019, 10, 30)

# ---------------------------------
# Main program
# ---------------------------------
def main() :
    # Make sure the data download dir exist
    makedirs(ARTICLES_DIR, exist_ok=True)

    # download_date ("2019-10-12", "testfil.json") - test document for
```

```
processing..
    download_date_range( START_DATE, END_DATE, ARTICLES_DIR)

# ---------------------------------------
# Iterate over each day and download the data
# ---------------------------------------

def download_date_range ( start_date, end_date, articles_dir ) :
    # REFERENCE: day iteration from here:
    # http://stackoverflow.com/questions/7274267/print-all-day-dates-
between-two-dates

    dayrange = range((end_date - start_date).days + 1)
    for daycount in dayrange:
        current_date = start_date + timedelta(days=daycount)
        current_datestr = current_date.strftime('%Y-%m-%d')
        print("\n---------- Processing: ", current_datestr)

        fname = join(articles_dir, current_datestr + '.json')
        if exists(fname):
            print("already downloaded, skipping....")
        else:
            # then let's download it
            print("Downloading", current_datestr)
            download_date (current_datestr, fname)

#---------------------------------------
# Download the data for a specific date and write it to a JSON file
#
# Sample URL :
#
# http://content.guardianapis.com/search?from-date=2016-01-02&
# to-date=2016-01-02&order-by=newest&show-fields=all&page-size=200
# &api-key=your-api-key-goes-here
# -----------------
def download_date( datestr, filename) :

    my_params = {
    'from-date': "", # leave empty, change start_date / end_date variables
instead
    'to-date': "",
    'order-by': "newest",
    'show-fields': 'all',
    'page-size': 200,
    'api-key': MY_API_KEY
}

    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'])
        # if there is more than one page
        total_pages = data['response']['pages']
        current_page += 1


    with open(filename, 'w') as f:
        print("Writing to", filename)

        # re-serialize it for pretty indentation
        f.write(json.dumps(all_results, indent=2))




# ---------------------------
# Now all functions have been defined. Run the main program.
main() # This should be the last line of the file !
```

preprocessing.py

- from https://github.com/BA-ODS2019/dgk432/blob/master/P2/preprocessing.py

```
#!/usr/bin/env python3
import json
import os

import pandas as pd
import numpy as np
import string
import re
from collections import Counter
import random

# Python Libraries for visualising
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Python Libraries for Tf-idf vectorizing
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import stopwords as sw
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.decomposition import LatentDirichletAllocation

# -------------
# CONSTANT DEFINITIONS
# ------------------

EXCLUDE_SECTION_IDS = {'culture', 'about', 'housing-network', 'fashion',
'travel', 'technology', 'australia-news', 'film',
'education', 'clinique-beyond-beauty', 'provoking-progress',
'commentisfree', 'books', 'theobserver',
'breakthrough-science', 'kids-travel-guides', 'crosswords', 'football',
'born-adventurous', 'sport', 'raising-cats-and-dogs',
'fill-your-heart-with-ireland', 'membership', 'tv-and-radio', 'stage',
'science', 'the-a-to-z-of-sleep',
'britain-get-talking', 'music', 'food', 'spains-secret-coast',
'cities', 'lifeandstyle', 'artanddesign', 'games', 'guardian-
masterclasses'}

DATADIR = str( os.getenv('OPENDATASCIENCE_DATADIR') )

#-----------------------------------------------
# Getting access to  my downloaded Guardian text corpus - incl full text
#   - collection_subdirs : Directory containting all datasets to be read
#   -  exclude_sectionIds :
```

```
# returns list of articles as strings from my dataset
#-----------------------------------------------
def get_all_articles_texts( collection_subdir, exclude_sectionIds) :
    # Update to the directory that contains the json files ( articles from
my fetched dataset)
    # Note the trailing /

    #include_sections = {'world', 'cities', 'money', 'media', 'community',
'global-development',
    # 'society', 'info', 'business', 'science', 'environment',
'technology', 'politics',
    # 'global', 'news', 'law', 'uk-news', 'us-news'}

    directory_name = DATADIR + "/" + collection_subdir  #
"$OPENDATASCIENCE_DATADIR/theguardian/collection/"
    print(directory_name)

    unique_sectionIds       = set()
    unique_articletypes     = set()
    excluded_articles_count = 0
    texts = list()
    for filename in os.listdir(directory_name):
        if filename.endswith(".json"):
            with open(directory_name + filename) as json_file:
                data = json.load(json_file)
                for article in data:
                    sectionId = article['sectionId']
                    if sectionId not in exclude_sectionIds:
                        #
                        unique_sectionIds.add(sectionId)
                        articletype = article['type']
                        unique_articletypes.add(articletype)
                        fields = article['fields']
                        text   = fields['bodyText'] if fields['bodyText']
else ""
                        # ids.append(id)
                        texts.append(text)
                    else:
                        excluded_articles_count += 1

    # print("Number of ids: %d" % len(ids))
    print("Number of included documents : %8d" % len(texts))
    print("Number of excluded documents : %8d" % excluded_articles_count )

    print("Unique sectionIDs:" )
    print( unique_sectionIds )
    print("Unique article type:" )
    print ( unique_articletypes )

    all_lengths = list()
    for article in texts:
        all_lengths.append(len(article))
```

```
    # ------------------------------------------------
    # How long are the texts?
    # Calculate total length in characters per class.
    # ------------------------------------------------

    print("Total sum of all characters in articles: %i" %
sum(all_lengths))
    print("MEAN size of articles: %d characters" % np.mean(all_lengths))


    return texts

#------------------------------------------------
# Count words in a list of strings
# INPUT:
#   list_of_data : Array of strings containing long texts
# OUTPUT:
#   Total number of words in all the data.
#------------------------------------------------
def word_count ( list_of_data):
    # word count (simple tokenization)
    # basic text analysis

    word_count = 0
    for text in list_of_data:
        words = text.split()
        word_count = word_count + len(words)

    return word_count

def unique_words ( list_of_data ):
    # word count plus the unique word count
    unique_words = set()
    for text in list_of_data:
        words = text.split()
        unique_words.update(words)

        # reference taken from https://www.programiz.com/python-
programming/set
        # unique_words = set(all_words)

    unique_word_count = len(unique_words)

    return unique_word_count

#------------------------------------------------
# Calculate average length of a list of text strings
# INPUT:
#   list_of_data : Array of strings containing long texts
# OUTPUT:
#   Average length in characters (int)
#------------------------------------------------
def average_length( list_of_data ):
    total = 0
```

```
    for text in list_of_data:
        total = total + len(text)

    number_of_data_elements = len(list_of_data)

    average_article_length = total / number_of_data_elements

    return average_article_length


#-----------------------------------------------
# Get top 10 words from a list of full-texts
# INPUT:
#     list_of_data: Array of strings containing the text to be counted
# OUTPUT:
#     list of strings, containing top 10 words
# PROCESS:
#     removes english stopwords - apply token_pattern (includes only word
containing 2 characters/letters or more, a-zA-Z or - )
#
# Codesnippets are inspired by class exercises +
https://liferay.de.dariah.eu/tatom/preprocessing.html#index-0
#-----------------------------------------------
def topwords( list_of_data ):

    model_vect = CountVectorizer(max_features=10000,
stop_words=sw.words('english'), token_pattern=r'[a-zA-Z\-][a-zA-Z\-]{2,}')

    # train and apply the model
    data_vect = model_vect.fit_transform( list_of_data )
    print('Shape: (%i, %i)' % data_vect.shape)

    # The resulting matrix is a sparse matrix.
    dense_count = np.prod(data_vect.shape)
    zeros_count = dense_count - data_vect.size
    sparsity = zeros_count / dense_count

    print("dense count ..........: %d" % dense_count )
    print("zeros_count ..........: %d" % zeros_count )
    print("sparsity of the matrix: %f" % sparsity )


    # Show part of the document-term count matrix
    # Give the indexes of the top-10 most used words.
    counts = data_vect.sum(axis=0).A1
    top_idxs = (-counts).argsort()[:10]

    # Give the words belonging to the top-10 indexes.
    # Tip: Use an inverted vocabulary to get the words that match your
indexes.
    inverted_vocabulary = dict([(idx, word) for word, idx in
model_vect.vocabulary_.items()])
    top_words = [inverted_vocabulary[idx] for idx in top_idxs]
    print("Top words: %s" % top_words)
```

```
    # Show the document vectors for 10 random documents.
    # Limit the vector to only the top-10 most used words.
    # Note: To slice a sparse matrix use A[list1, :][:, list2]
    # (see https://stackoverflow.com/questions/7609108/slicing-sparse-
scipy-matrix)

    some_row_idxs = random.sample(range(0, len(list_of_data)), 10)
    print("Selection: " % (some_row_idxs))
    sub_matrix = data_vect[some_row_idxs, :][:, top_idxs].todense()
    print(sub_matrix)

    df = pd.DataFrame(columns=top_words, index=some_row_idxs,
data=sub_matrix)
    print("print the DataFrame - incl the random sample from the index")
    print(df)

    return top_words

# -------------------
# Search for a list of query terms in a list of full texts
# Use Term Frequency (tf-idf) indexing to return most relevant articles
# INPUT:
#   query : String with space separated query terms
# OUTPUT:
#     list of strings, containing the full texts that match the query
# -------------
def search_for( query, list_of_data ) :
    list_of_result_data = []

    model_vect = CountVectorizer(max_features=10000,
stop_words=sw.words('english'), token_pattern=r'[a-zA-Z\-][a-zA-Z\-]{2,}')
    data_vect = model_vect.fit_transform(list_of_data)

    # Apply TF-IDF weighting for this query model
    model_tfidf = TfidfTransformer()
    data_tfidf = model_tfidf.fit_transform(data_vect)

    query_vect_counts = model_vect.transform([query])
    query_vect = model_tfidf.transform(query_vect_counts)
    print("Query vector : ", query_vect)

    # Calculate the similarity between the query vector and each document.
    # Tip: Use scikit's cosine_similarity to compare the query vector with
the document-term tfidf matrix.
    sims = cosine_similarity(query_vect, data_tfidf)
    print("Cosine similarity : ", sims)

    sims_sorted_idx = (-sims).argsort()
    sims_sorted_idx # Article indexes sorted by similarity to the query

    top_article_text = list_of_data[sims_sorted_idx[0,0]]
    print("Top article matching my query : ", top_article_text)
```

```
    # Tabulate top-10 document indexes and similarities using pandas
DataFrame
    # Tip: cosine_similarity returns a nested (query x documents) array
("sims").
    # Evaluate len(sims) and len(sims[0,:])) first.
    print("Shape of 2-D array sims: (%i, %i)" % (len(sims),
len(sims[0,:])) )
    df = pd.DataFrame(data=zip(sims_sorted_idx[0,:],
sims[0,sims_sorted_idx[0,:]]), columns=["index", "cosine_similarity"])
    print("Dataframe with top 10 document indexes : ", df[0:10])

    print("Getting top 10 articles ....")
    for idx in sims_sorted_idx[0,0:10] :
        article_text = list_of_data[idx]
        print( idx )
        #print( article_text )
        list_of_result_data.append( article_text  )

    return list_of_result_data


# --------------------
# Search for top 10 words in a list of article texts and create a
WordCloud with ...
# Use model vector and LatentDirichletAllocation
# INPUT:
#   query : String with space separated query terms
# OUTPUT:
#     list of strings, containing the full texts that match the query
# -------------
def create_topics_wordcloud( list_of_data, title, filename ) :

    model_vect = CountVectorizer(max_features=10000,
stop_words=sw.words('english'), token_pattern=r'[a-zA-Z\-][a-zA-Z\-]{2,}')
    data_vect = model_vect.fit_transform(list_of_data)

    print("Random names: ", random.sample(model_vect.get_feature_names(),
10) )

    # Perform topic modelling
    # using Latent Dirichlet Allocation (LDA) on your pre-processed data
    # Note: Set n_components=4 (nr. of topics) and random_state=0 (so we
get..... #TODO:
    model_lda = LatentDirichletAllocation(n_components=4, random_state=0)
    data_lda = model_lda.fit_transform(data_vect)

    # Describe the shape of the resulting matrix.
    import numpy as np
    np.shape(data_lda)

    # Display the top-10 terms and their weights for each topic.
    # Are the topics recognisable?
    # Tip: Use the components_ attribute of your LDA model, which contains
the topic-t
```

```
    # Use enumerate to loop over the components.
    for i, term_weights in enumerate(model_lda.components_):
        top_idxs = (-term_weights).argsort()[:10]
        top_words = ["%s (%.3f)" % (model_vect.get_feature_names()[idx],
term_weights[idx]) for idx in top_idxs]
        print("Topic %d: %s" % (i, ", ".join(top_words)))
    # A big challenge of topic modelling is too make the topics
interpretable and meaningfull

    # Display doc-topic matrix for 10 documents
    topic_names = ["Topic" + str(i) for i in
range(model_lda.n_components)]
    doc_names = ["Doc" + str(i) for i in range(len(list_of_data))]
    df = pd.DataFrame(data=np.round(data_lda, 2), columns=topic_names,
index=doc_names).head(10)
    # extra styling
    #df.style.applymap(lambda val: "background: yellow" if val>.3 else '',
)
    print("dataframe: ", df)

    # Inspect a random document
    doc_idx = random.randint(0,len(list_of_data)-1)
    print('Doc idx: %d' % doc_idx)
    # What is the most likely topic according to LDA?
    topics = data_lda[doc_idx]
    print('Topic vector: %s' % topics)
    vote = np.argsort(-topics)[0]
    print('Topic vote: %i' % vote)
    # Look at the textual content of the document
    print('Document text: ', list_of_data[doc_idx])

    # Topic visualization by use of Wordcloud
    # Word cloud
    # Display a word cloud for a topic. Display the top-10 words and use
their term loadings as weight/frequency.
    # Tip: Install and use the wordcloud package.
    # Example usage: https://stackoverflow.com/questions/43145199/create-
wordcloud-fro.....#TODO

    plt.clf()

    for i, term_weights in enumerate(model_lda.components_):
        top_idxs = (-term_weights).argsort()[:10]
        top_words = [model_vect.get_feature_names()[idx] for idx in
top_idxs]
        word_freqs = dict(zip(top_words, term_weights[top_idxs]))
        wc = WordCloud(background_color="white",width=300,height=300,
max_words=10).generate_from_frequencies(word_freqs)
        plt.subplot(2, 2, i+1)
        plt.imshow(wc)
        plt.axis("off")

    plt.suptitle(title)
    plt.savefig(filename)
```

```
    return

#-------------------------------------------------------------------
# MAIN PROGRAM ...
#-------------------------------------------------------------------

# Update to the directory that contains your json files
# Note the trailing /

do_not_exclude_anything= set()

# 1. Get the data set .............
# below are some of my test-data-sets for fetching data - since the data
range is very large, I have
# selected a few articles in a test collection for the text analysis / pre
processing for increasing
# speed
#list_of_articlefulltexts =
get_all_articles_texts("theguardian/collection_test/",
EXCLUDE_SECTION_IDS)
#list_of_articlefulltexts =
get_all_articles_texts("theguardian/collection/", do_not_exclude_anything
)
list_of_articlefulltexts =
get_all_articles_texts("theguardian/collection_full/",
EXCLUDE_SECTION_IDS)


# 2. analyse and visualize full data set ......................

article_top_words = topwords( list_of_articlefulltexts )
word_count =  word_count( list_of_articlefulltexts)
unique_word_count = unique_words( list_of_articlefulltexts)
average_article_length = average_length( list_of_articlefulltexts  )

print("word_count: ........... %9i" % word_count)
print("Unique word count: .... %9i" % unique_word_count)
print("Average article length: %9i" % average_article_length)

# 3. Search for ... Query: with which topics was "Bill Clinton ... "
discussed? ...................

bill_clinton_articles = search_for( "Bill Clinton",
list_of_articlefulltexts)
create_topics_wordcloud(bill_clinton_articles, "Bill Clinton in
Wordclouds", "clinton_wordcloud.png")

trumps_articles = search_for( "Donald Trump", list_of_articlefulltexts)
create_topics_wordcloud(trumps_articles, "Donald Trump in Wordclouds",
"trump_wordcloud.png")

climate_articles = search_for( "Climate change", list_of_articlefulltexts)
create_topics_wordcloud(climate_articles, "Climate Change Topics in
```

```
Wordclouds", "climate_wordcloud.png")

thunbergs_articles = search_for( "Greta Thunberg",
list_of_articlefulltexts)
create_topics_wordcloud(thunbergs_articles, "Greta Thunberg in
Wordclouds", "thunberg_wordcloud.png")


# --------------
```

Portfolio 3 handler om datasæt fra SMK.

Ad spm. 1).

Spørgsmål 1 i afleveringen handler om at beskrive baggrunden for, hvorfor SMK har valgt at digitalisere deres samling og hvad de har gjort sig af overvejelser om, hvad disse åbne SMK datasæt kan anvendes til..

SMK har valgt at stille deres samling til rådighed via en åben API, som ikke kræver registrering for at få en "personlig" API KEY. Man kan blot via en *simpel søgegrænseflade* fremsøge og hente relevant data/metadata til studie, forskning og almindelig nysgerrighed.

Projektet har SMK valgt at kalde "SMK OPEN" og blev startet i 2016 med funding fra bl.a. NORDEA fonden. Digitaliseringen af de mange kunstværker har til formål at give adgang og åbne op for kunst, så mange flere kan få glæde af kunsten. I blog-indlægget(1) beskriver de lidt om bevæggrundene for, hvorfor de har taget dette initiativ. Dels for at synliggøre kunst for alle, også skoleklasser som eksempelvis ikke lige rejser mange timer gennem Danmark for at se "udstillet materiale", men primært for at få tilgængeliggjort og synliggjort kunstværker, da "langt størstedelen af kunstsamlingen opbevares i magasiner og sjældent eller aldrig ser dagens lys. I magasinerne bor kunsten trygt og godt. Der bliver passet på den. Men hvad er meningen med al den kunst, hvis ingen får glæde af den?".

Meningen er, at det digitaliserede materiale/museets kunstværker kan hentes ned på ens computer, og man vil kunne bruge dem til lige det, man har lyst til.

SMK begrunder selv, hvorfor de har besluttet at stille data til rådighed for alle deres digitaliserede værker via API således - "Foruden præsentation på hjemmesiden vil al SMK's data blive tilgængeliggjort via et såkaldt API. Hermed kan data bruges direkte af andre digitale tjenester (hjemmesider, apps osv.) så eksterne udviklere kan bygge digitale tjenester som trækker direkte på SMK's data.deres API adgang ud fra ".

Een APP som allerede er et resultat af disse åbne datasæt er VIZGU(2) der gør det lettere for besøgende på SMK at navigere i kunstværkerne, som udstilles. Man anvender APP´en, scanner lidt af det "udstillede" og får et svar retur med beskrivelser og supplerende information om værket man ser.

SMK er ikke alene om at stille deres samlinger til rådighed - flere offentlige organisationer verden over sørger for at få digitaliseret materiale, samlinger og andet til gavn for offentligheden, for forskningen og for forståelse af den kultur vi lever i. I artiklen fra Medium beskriver Merete Sanderhoff netop, hvilke overvejelser man tænker modtagerne kan anvende disse data til: "This is opening up new powerful opportunities for citizens to engage with, study, repurpose and remix large sets of high quality trustworthy data collected by public institutions over many years."(3) og i den officielle blog-post fra Jonas Heide Smith(4) skriver han netop om målet som "make the art collection of the Danish people actually usable. In a way that as many people as possible will be able to enjoy."

SMKs digitale værker er i store træk indekseret via en form for kunstig intelligens. Denne proces omtales som "algorithm gnomes" - fantastisk ord :-) Desuden har alle kunstværker, skitser mv. fået tilknyttet relevant metadata, favnende fra størrelser på lærreder til farve-toner, hvor og hvornår værket er udstillet og hvilken dokumentation/litteratur som knytter sig til værkerne eller kunstneren.

Ideen med API adgangen er netop, at data kan tilgåes uafhængigt af tid og rum, kan bearbejdes ud fra egne behov og ønsker, kan studeres i detaljer og bare i rette sammenhæng, kan deles, indsættes i alt fra bøger, websider, forskningsartikler til skoleopgaver, eller billeder kan genoptrykkes til alt fra plakater til sofapuder(5). Ifølge den officielle projektbeskrivelse har SMK 2 konkrete målgrupper i tankerne - dels skolelærere, men også DIY folket (6). Skolelærerne for at få "formidlet kunst og dannet børn og unge omkring vores fælles kulturarv, mens DIY folket, kreative sjæle og andre med interesse for at undersøge, genanvende og videreudvikle kunst

værker i "nye klæder" og til andre sjove formål, herunder at få Matisse på madkassen eller Vilhelm Hammershøi på kaffekoppen. Når billeder fundet via SMK Open er "knyttet til en *public domain* licens", så kan de i princippet anvendes frit.

1. https://www.smk.dk/article/kunsten-vil-gerne-vaere-fri/ (https://www.smk.dk/article/kunsten-vil-gerne-vaere-fri/)
2. https://www.smk.dk/article/vizgu/ (https://www.smk.dk/article/vizgu/)
3. https://medium.com/smk-open/smk-open-nominated-for-open-data-award-2018-e4b46e17bcf (https://medium.com/smk-open/smk-open-nominated-for-open-data-award-2018-e4b46e17bcf)
4. https://medium.com/smk-open/were-open-thoughts-on-building-a-new-home-for-smk-s-online-collection-9c9adbbe08a4 (https://medium.com/smk-open/were-open-thoughts-on-building-a-new-home-for-smk-s-online-collection-9c9adbbe08a4)
5. https://open.smk.dk/about (https://open.smk.dk/about)
6. https://www.smk.dk/wp-content/uploads/2018/06/Projektbeskrivelse_SMK-Open.pdf (https://www.smk.dk/wp-content/uploads/2018/06/Projektbeskrivelse_SMK-Open.pdf)

Ad spm. 2)

SMK data hentes via https://www.smk.dk/article/smk-api/ (https://www.smk.dk/article/smk-api/)

Jeg har valgt at søge på *Vilhelm Hammershøi* via GET -> art/search og ved at anvende feltet KEYS - samt sætte viste ROWS til 200. Resten lod jeg være standard. Min json fil bliver derfor vist som SMK JSON format.

URL (resultatet ses som een del af output fra søgning via ovenstående) og ser således ud https://api.smk.dk/api/v1/art/search/?keys=vilhelm%20hammersh%C3%B8i&offset=0&rows=200 (https://api.smk.dk/api/v1/art/search/?keys=vilhelm%20hammersh%C3%B8i&offset=0&rows=200)

Der er i alt 104 hits, som matcher min *query* - og jeg har tjekket Metadata elementerne (struktureret datasæt), som beskriver de respektive hits via *Visual Studio Code*, som har en JSON beautifier add on.

En JSON fil vises lidt anderledes end en flad csv fil. Python biblioteket *pandas* har en funktion/metode *read_json som gør det muligt at loade en .json file ind i en *pandas dataframe*, men uanset denne funktion, så efterlader json filen os stadig med mulige lister/dictionaries i de enkelte kolonner. Dvs. der er forskellige niveaer i fremvisningen (træ-struktur), hvilket betyder, at man er nødt til at "flade json filen en anelse" for at kunne arbejde med ens data i pandas, som er bedst til tabel-struktur. Derfor er det altid smart at tjekke data igennem, så man kan danne sig et overblik over, hvilke felter, der findes og hvilke felter, det er værd at undersøge nærmere på, jv. https://www.kaggle.com/tboyle10/working-with-json-files (https://www.kaggle.com/tboyle10/working-with-json-files) og https://towardsdatascience.com/flattening-json-objects-in-python-f5343c794b10 (https://towardsdatascience.com/flattening-json-objects-in-python-f5343c794b10)

a) *Typer metadata* som findes i resultatet

I det resultat (104 hits) som jeg får via min *query* kan jeg se nogle over-kategorier [offset, rows, found, items, facets, facet_ranges, corrections or autocomplete] Jeg har valgt at kigge nærmere på indholdet af kategorien **items**, hvorfra følgende (meta)data elementer forefindes:

ID created & modified acquisition date responsible department current locaation name content description content person disinguishing features frame notes dimensioner (på rammen/værket) documentation (herunder nævnes/samlet litteraturen som beskriver værket - inkl. forfatter, title, hyldeplacering..) exhibitions (steder hvor værket har været udstillet og hvornår.) inscriptions object name object history note production -> som bl.a. indeholder *creator* / data om ophav production date notes work status techniques titler (inkl.

language, type..) object number object URL rights (public domain) on display .. info om image (has image, link til image, fakta om image..) colors (et søgeparameter, som især er unikt fra den åbne søgefunktion via SMK Open..)

---

b) *SMK (meta)data struktur versus Dublin Core*

Inspireret fra pensum litteraturen(1) samt tilhørende overblik fra web (2) er SMK datasættet blevet sammenlignet med **Dublin Core metadata** formatet for at finde evt. overensstemmelser. Ideen med at få beskrevet data ved hjælp af metadata er, at man så kan genfinde, genskabe, skabe hoved og hale, samt få tilstrækkelig indsigt i data. Uden god og fornuftig brug af metadata (katalogisering, videnorganisering) vil digitale objekter (billeder, artefakter, bøger, artikler, information, video, software, datasæt..) være uden mening og samlingerne vil være ubruglige/usynlige og umulige at finde igen. Metadata er i bund og grund blot "data about data", ... "a means by which largely meaningless data may be transformed into information, interpretable and reusable by those other than the creator of the data resource"(1). Men metadata er også med til at beskrive data, som måske IKKE må være åbne for alle med det samme. Disse data kan medvirke til at gøre data FAIR - så overblik, indsigt, potentiel nytte kan opnås - så man ved hvem der er ophav, hvilke formater data er i, hvor det er gemt, hvordan man får adgang, hvordan det er knyttet til anden data osv. Og så kan god Metadata mappes til andre standarder, så data kan fra eet sted (eks. SMK) kan anvendes i andre systemer og søge-kontekster. jv. spm hvor SMK struktur sammenlignes m. Dublin Core(DC).

DC er et internationalt anerkendt standardformat opstået helt tilbage i 1995, da internettet stadig var ungt og nyt. Kernen af metadatafelter består af 15 elementer, i første omgang opstået for at kunne beskribe web baserede ressourcer fra eksempelvis "kulturarven fra GLAM" (Galleries, Libraries, Arhives and Museums). DC udvikler sig stadig og hvis man er interesseret i indeksering og metadata, så er https://dublincore.org/ (https://dublincore.org/) et godt udgangspunkt for mere undersøgelse.

| **SMK (meta)data struktur** | **Simple Dublin Core elements** |

SMK har nogle felter til beskrivelse af værket: DC har lignende felter som kan mappes til SMK elementerne:

ID, dato for oprettelse af posten(created), Date: omfatter SMKs (created, anskaffelse, indgået i sam-anskaffelsesdato (Acquisition_date), hvor værket ling osv) findes (responsible_department), content_person, Format: fil-format, dimensioner dimensioner på værket, documentation (inkl. litt), Source: svarer lidt til SMKs Documentation som handler om materialet som værket er på, licens/brug/kreditering relatede kilder Rights: omfatter både licens og brugsret

SMK vælger felter/elementer som beskriver værker som malerier, tegninger, skitser, mens DC er tænkt som et standardformat bredt og generisk nok til "use in a cross-disciplinary information environment"(1) og derfor har DC felter som *title, creator, description.abstract, publisher, subject.. mv.*" som så vil skulle oversættet til SMK´sk med felter som *titles - på værket, creator - til værket, ... *, mens publisher, subject osv skal findes via de beskrivende felter fra nogle af de nævnte eksempler ovenfor.

1. Monson, J. D. (2017). Getting started with digital collections: scaling to fit your organization. Chicago: ALA Editions, an imprint of the American Library Association. (heruder kapitel 6 om **Metadata**)
2. https://dublincore.org/resources/userguide/ (https://dublincore.org/resources/userguide/)

c) *Tal (data) fra resultatet* som evt. kan anvendes til statistiske beregninger (ex årstal)

SMK beskrivelser som inkl tal er primært at finde via felter som *created and modified*, samt *acquisition date*, *exhibition dates* - men også størrelser på værkerne kan sammenlignes.

d) Eksempler på, hvilke slags *spørgsmål man kan besvare gennem analyse* af dette datasæt

Det er oplagt at undersøge, hvornår de mange værker er anskaffet til SMK/dvs. tidspunktet/dato (feltet som det undersøges på er *acquisition date"). Een af de andre spørgsmål man kan undersøge er, hvor store er de værker som har været udstillet i et givent venue - eller er nogle teknikker, som eksempelvis 'tegning' ikke udstillet i SMK/Sølvgade.

Jeg har dog valgt at begrænse ambitionerne og valgt at kigge på og visualisere følgende:

a. I hvilken "time-range" er værkerne anskaffet (årstal, som kan findes via at slice nogle lister fra en overordnet DataFrame b. Hvor mange værker (i procent) har været udstillet (til de udarbejdes en ekstra exhibition DataFrame)

Løsningen til Portfolio 3 fra spm 3 og frem kræver brug af Jypyter Notebook, som er et nyt Python tool, men som i store træk ligner Spyder/Anaconda, som jeg har anvendt i Portfolio 1 og 2.

Man skal stadig have **importeret** de nødvendige moduler til sin Jypyter notebook - jeg har hentet følgende for at kunne løse spørgsmålene:

- import pandas - normally as pd
- from pandas import DataFrame
- from pandas.io.json import json_normalize
- import json normalize module - because json requires a little extra
- selve json filen via json
- import numpy - again the default alias being np
- for visualisation - import matplotlib as plt
- samt requests module

In [1]:

```
import requests
import json
import pandas as pd
import numpy as np
from pandas.io.json import json_normalize #package for flattening json in pandas df
from pandas import DataFrame
import matplotlib.pyplot as plt
```

Hente data

Jeg har valgt at søge på *Vilhelm Hammershøi* via GET -> art/search og ved at anvende feltet KEYS - samt sætte antal ROWS til 200. Resten lod jeg være standard.

Mit resultat (*.json fil) bliver derfor vist som SMK JSON format.

In [2]:

```
# a variable for the API URL
api_search_url = 'https://api.smk.dk/api/v1/art/search/'

# a dictionary called 'params' setting the parameters I used for searching the API
params = {
'keys': 'vilhelm hammershøi' ,
'rows': '200'

}

response = requests.get(api_search_url, params)
json_data = response.json()

print(json.dumps( json_data, sort_keys=True, indent=4))
```


```
{
    "autocomplete": [
        "Hammersh\u00f8i, Vilhelm"
    ],
    "corrections": [],
    "facets": {},
    "facets_ranges": {},
    "found": 104,
    "items": [
        {
            "acquisition_date_precision": "1971-12-31",
            "alternative_images": [
                {
                    "height": 5538,
                    "iiif_id": "https://iip.smk.dk/iiif/jp2/KMS6692.t
if.jp2",
                    "iiif_info": "https://iip.smk.dk/iiif/jp2/KMS669
2.tif.jp2/info.json",
                    "mime_type": "image/tiff",
                    "native": "https://iip.smk.dk/iiif/jp2/KMS6692.ti
```


In [3]:

```
print(len(json_data))
# tager antal af niveau 1 i json filen
# - inkl felter som "offset", "rows", "found", "items", etc.
```

```
8
```

In [4]:

```
print(len(json_data['items']))
# printer antal resultater ud, som matcher antallet i søgningen = 104 hits

# - da alle mine resultater har en beskrivelse under facetten 'items'
```

```
104
```

...via Python biblioteket json_normalize burde jeg kunne "normalisere mit dataset (json_data)" med henblik på at kunne finde ind til det niveau, som jeg ønsker at undersøge nærmere. Jeg har bokset en del med det, og tænker, at for at kunne køre denne code korrekt, skal alle de kørte resultater (104 hits) indeholde en værdi eller blot medtage den "undersøgende facet", hvis ikke det er tilfældet (som i mit tilfælde), så skal man kode sig til

en løsning Dokumentation er læst op via https://github.com/pandas-dev/pandas/blob/v0.25.3/pandas/io/json/_json.py#L334-L598list (https://github.com/pandas-dev/pandas/blob/v0.25.3/pandas/io/json/_json.py#L334-L598list)

...nedenfor har jeg i første omgang kreeret en Pandas DataFrame, som vil danne grundlag for mine undersøgelser af datasættet. 'items' synes at være den rette over-facet at dykke ind i. En basis data analyse foretages via lært metode..

In [5]:

```
all_df = pd.DataFrame(json_data['items'])

print(all_df.dtypes)
print(all_df.shape)
print(all_df.size)

all_df.get_dtype_counts()
```

```
acquisition_date               object
acquisition_date_precision     object
alternative_images             object
colors                         object
content_description            object
content_person                 object
content_subject                object
created                        object
current_location_name          object
dimensions                     object
distinguishing_features        object
documentation                  object
exhibitions                    object
frame_notes                    object
has_image                        bool
id                             object
iiif_manifest                  object
image_cropped                  object
image_height                  float64
image_iiif_id                  object
image_iiif_info                object
image_mime_type                object
image_native                   object
image_orientation              object
image_size                    float64
image_thumbnail                object
image_type                     object
image_width                   float64
inscriptions                   object
labels                         object
materials                      object
modified                       object
notes                          object
number_of_parts               float64
object_history_note            object
object_names                   object
object_number                  object
object_url                     object
on_display                       bool
part_of                        object
parts                          object
production                     object
production_date                object
production_dates_notes         object
public_domain                    bool
related_objects                object
responsible_department         object
rights                         object
techniques                     object
titles                         object
work_status                    object
```

```
dtype: object
(104, 51)
5304
```

Out[5]:

```
float64     4
bool        3
object     44
dtype: int64
```

Datasættet (DataFrame på dette 'items' niveau) består af object (strings), float (numbers) og bool(True/False)

I alt 104 rækker, fordelt på 51 kolonner. Hvorvidt der er nogle kolonner som jeg IKKE kan bruge handler måske mest om kendskab til indhold i data og den kontekst man tænker det skal bruges i, så mit umiddelbare svar er at alle kolonner kunne være relevante for en undersøgelse, men det er tiden ikke til i denne opgave..

.. nedenfor vises de 5 første poster fra min DataFrame.

Jeg kan se, at nogle af mine hits ikke har nogle værdier (NaN indikerer Not a Number), Værdien vises imidlertid i min DataFrame fra Pandas, så har jeg ikke nogle "tomme felter", men jeg vil dels kunne sortere på NaN værdien, erstatte den, fjerne den eller lignende - kilden til den metode/viden findes i Pandas Dokumentationen: https://github.com/pandas-dev/pandas/blob/v0.25.3/pandas/core/base.py#L1304-L1394 (https://github.com/pandas-dev/pandas/blob/v0.25.3/pandas/core/base.py#L1304-L1394) - se mere i slutningen af dette dokument, hvor jeg visualiserer på denne i procent..

In [6]:

```
print(all_df.head())
```

```
  acquisition_date acquisition_date_precision  \
0              NaN                  1971-12-31
1              NaN                  1916-12-31
2              NaN                  1997-12-31
3              NaN                  1972-12-31
4              NaN                  1924-12-31

                                alternative_images  \
0  [{'mime_type': 'image/tiff', 'iiif_id': 'https...
1  [{'mime_type': 'image/tiff', 'iiif_id': 'https...
2  [{'mime_type': 'image/tiff', 'iiif_id': 'https...
3  [{'mime_type': 'image/tiff', 'iiif_id': 'https...
4  [{'mime_type': 'image/tiff', 'iiif_id': 'https...

                                      colors content_description
\
0  [#555555, #555555, #000000, #5A5A5A, #9B9B9B]                NaN
1  [#555555, #555555, #555555, #5A5A5A, #9B9B9B]                NaN
2  [#9B9B9B, #9B9B9B, #5A5A5A, #F4C6A3, #FFFFFF]                NaN
3  [#9B9B9B, #5A5A5A, #555555, #9B9B9B, #555555]                NaN
4  [#9B9B9B, #D2D2D2, #5A5A5A, #555555, #D2D2D2]                NaN

          content_person        content_subject              created
\
0  [Hammershøi, Vilhelm]                    NaN  2019-08-07T03:43:03Z
1  [Hammershøi, Vilhelm]                [Virum]  2019-08-07T03:21:50Z
2  [Hammershøi, Vilhelm]                    NaN  2019-08-07T05:14:58Z
3                    NaN  [Christiansborg Slot]  2019-08-07T07:22:45Z
4                    NaN       [Christianshavn]  2019-08-07T05:58:07Z

  current_location_name                                          dimens
ions  \
0              Sal 228  [{'notes': '645 x 514 x 75 mm', 'part': 'brut
t...
1              Sal 228  [{'notes': '1260 x 1495 mm', 'part': 'netto',
...
2                  NaN  [{'notes': '247 x 189 mm', 'part': 'bladmaa
l',...
3              Sal 228  [{'notes': '1367 x 1700 x 110 mm', 'part': 'b
r...
4              Sal 228  [{'notes': '639 x 702 x 55 mm', 'part': 'brut
t...

   ...                                          production  \
0  ...  [{'creator': 'Hammershøi, Vilhelm', 'creator_d...
1  ...  [{'creator': 'Hammershøi, Vilhelm', 'creator_d...
2  ...  [{'creator': 'Hammershøi, Vilhelm', 'creator_d...
3  ...  [{'creator': 'Hammershøi, Vilhelm', 'creator_d...
4  ...  [{'creator': 'Hammershøi, Vilhelm', 'creator_d...

                                   production_date  \
0  [{'start': '1890-01-01T00:00:00.000Z', 'end': ...
1  [{'start': '1911-01-01T00:00:00.000Z', 'end': ...
2  [{'start': '1898-01-01T00:00:00.000Z', 'end': ...
3  [{'start': '1890-01-01T00:00:00.000Z', 'end': ...
4  [{'start': '1901-01-01T00:00:00.000Z', 'end': ...
```

```
                              production_dates_notes public_domain  \
0          [Datering if. Bramsen & Michaëlis 1918  ]          True
1           [Datering if. Bramsen og Michaëlis 1918]          True
2                                                NaN          True
3  [Maleriet blev ifølge Michaëlis & Bramsen påbe...          True
4            [Datering if. Bramsen & Michaëlis 1918]          True

   related_objects      responsible_department  \
0              NaN  Samling og Forskning (KMS)
1              NaN  Samling og Forskning (KMS)
2              NaN  Samling og Forskning (KKS)
3              NaN  Samling og Forskning (KMS)
4              NaN  Samling og Forskning (KMS)

                                              rights  \
0  https://creativecommons.org/share-your-work/pu... (https://creative
commons.org/share-your-work/pu...)
1  https://creativecommons.org/share-your-work/pu... (https://creative
commons.org/share-your-work/pu...)
2  https://creativecommons.org/share-your-work/pu... (https://creative
commons.org/share-your-work/pu...)
3  https://creativecommons.org/share-your-work/pu... (https://creative
commons.org/share-your-work/pu...)
4  https://creativecommons.org/share-your-work/pu... (https://creative
commons.org/share-your-work/pu...)

                                          techniques  \
0                 [{'technique': 'Olie på lærred'}]
1                 [{'technique': 'Olie på lærred'}]
2  [{'technique': 'blyant på papir linieret på fo...
3                 [{'technique': 'Olie på lærred'}]
4                 [{'technique': 'Olie på lærred'}]

                                              titles work_status
0  [{'title': 'Selvportræt', 'language': 'da-DK',...         NaN
1  [{'title': 'Selvportræt. Spurveskjul', 'langua...         NaN
2  [{'title': 'Selvportræt, set bagfra', 'languag...         NaN
3  [{'title': 'Fra det gamle Christiansborg. Sent...         NaN
4  [{'title': 'Stue i Strandgade med solskin på g...         NaN

[5 rows x 51 columns]
```

In [7]:

```
all_df["acquisition_date_precision"]
# visning af en konkret kolonne
#en kolonne som ikke består af nye indeholdte lister eller dictionaries,
# men blot een værdi ( dato eller NaN )
```

Out[7]:

```
0      1971-12-31
1      1916-12-31
2      1997-12-31
3      1972-12-31
4      1924-12-31
5      1928-12-31
6      1970-12-31
7      1970-12-30
8      1916-10-30
9      2005-05-13
10     2008-11-06
11     1998-09-10
12     1916-12-31
13     1924-12-31
14     1984-05-02
15     1908-12-31
16     1974-12-31
17     2012-09-17
18     2015-09-27
19     1938-12-31
20     1904-01-15
21     1988-03-24
22     1971-12-31
23     1924-12-31
24     1896-12-31
25     1916-12-31
26     1908-12-31
27     1926-02-16
28     1912-12-31
29     1970-12-31
          ...
74     1988-12-31
75     1992-09-02
76     2010-03-22
77     2016-04-18
78     1948-12-31
79     1964-12-31
80            NaN
81            NaN
82            NaN
83     2006-08-23
84     1948-12-31
85            NaN
86     1971-12-31
87            NaN
88     1998-10-01
89     2011-11-04
90     1918-12-31
91     1975-06-10
92     1955-04-26
93            NaN
94            NaN
```

```
95     1986-12-31
96            NaN
97     1914-12-31
98            NaN
99     1915-03-27
100    1899-12-31
101    1932-12-31
102    1955-05-04
103    1981-12-30
Name: acquisition_date_precision, Length: 104, dtype: object
```

Når man tjekker sit *.json datasæt igennem kan man se i formatet at enkelte indførsler består af både en liste og en data dictionary. Dvs. man skal arbejde med at hente data ud fra "flere niveauer" fra json strukturen.

Nedenunder har jeg hentet de 10 første indførsler fra DataFrame´ns **documentation** loc statement(location) og man kan se at der skal graves en anelse :

....fra "documentation" -> {'title': 'Hammershøi: værk og liv', 'author'.....} så hvis man skal arbejde med data fra denne dictionary skal der lidt python magi og snilde i spil.

In [8]:

```
all_df.loc[0:, "documentation"].head(10)
# denne dataframe indeholder et array/liste af data dictionaries på hele datasættet
```

Out[8]:

```
0    [{'title': 'Svend Hammershøi – en kunstner og ...
1    [{'title': 'Hammershøi: værk og liv', 'author'...
2    [{'title': 'Vilhelm Hammershøi: På sporet af d...
3    [{'title': 'Den forunderlige stilheten: Ida Lo...
4    [{'title': 'Den forunderlige stilheten: Ida Lo...
5    [{'title': 'Hammershøi: værk og liv', 'author'...
6    [{'title': 'Vilhelm Hammershøi and Danish art ...
7    [{'title': 'The  soul of the North: a social, ...
8    [{'title': 'Afstandens mellemværende: iscenesæ...
9    [{'title': 'De gouden eeuw in perspectief: het...
Name: documentation, dtype: object
```

Mit søgeresultat fra SMK API'en er også gemt lokalt, så jeg kan gennemtjekke elementer, og se data igennem lokalt - efter behov. Det kan lade sig gøre, fordi datasættets enkelte indførsler meget ligner hinanden, og fordi datasættet IKKE er enormt stort.

I koden nedenfor, hvor jeg forsøger at stikke snablen ned i en understruktur som exhibitions netop ved at anvende json_normalize funktionen får jeg en fejl. Ved at google lidt rundt viser det sig at "andre deler den samme oplevelse, og argumentet er følgende: Nogle af de fundne poster fra mit datasæt synes ikke at have indførslen **exhibitions** og derfor fejler dette.. ØV.

Men det er jo også en øvelse i preprocessering af data - nemlig at teste det for evt. mangler og andet godt..

Og!! Derfor lod jeg med vilje fejl-koden blive i denne aflevering, som eksempel på, at hvis data IKKE er så godt som man forventede, så vil fejl opstår, og fejlfinding har vist sig at være een stor del af "programmeringens vidunderlige verden"

man får en KeyError, sikkert fordi *exhibition feltet* ikke er udfyldt i hele datasættet..

In [9]:

```
# exhibition_df =json_normalize(json_data['items'],
# record_path="exhibitions", meta=['id'], meta_prefix="item.", errors='ignore')
```

... en *.json fil skal derfor igennem flere iterationer

Men vha Pandas DataFrame, så kan denne proces lettes - den kan arbejdes med i Python - dette kaldes "flattening" (link https://towardsdatascience.com/flattening-json-objects-in-python-f5343c794b10fået (https://towardsdatascience.com/flattening-json-objects-in-python-f5343c794b10f%C3%A5et) fra underviser er anvendt nedenfor - men viste sig lidt uoverkommeligt..

Anden løsning vælges - nemlig at lave 2 ekstra DataFrames konkret på exhibitions og documents

Ved at anvende "def flatten_json funktionen" så kan man flade sit datasæt ud og få overblik over, hvor mange og lange indeholdte lister og dictionaries fra kolonne værdierne er. Texten fremstår nu som en kæmpe lang flad fil, som der kan arbejdes på.

Hver enkelt indførsel tildeles egen unikke selvstændige indførsel, med en form "for katalog ID nummerering knyttet til kolonnens indhold (liste indhold/dictionary indhold)" så kan kan se antal og data fra indholdet i kolonnen.

In [10]:

```
def flatten_json(y):
    out = {}

    def flatten(x, name=''):
        if type(x) is dict:
            for a in x:
                flatten(x[a], name + a + '_')
        elif type(x) is list:
            i = 0
            for a in x:
                flatten(a, name + str(i) + '_')
                i += 1
        else:
            out[name[:-1]] = x

    flatten(y)
    return out
```

In [11]:

```
print(flatten_json(json_data))
```

```
{'offset': 0, 'rows': 200, 'found': 104, 'items_0_id': '1180006281_ob
ject', 'items_0_created': '2019-08-07T03:43:03Z', 'items_0_modified':
'2019-08-12T09:23:28Z', 'items_0_acquisition_date_precision': '1971-1
2-31', 'items_0_responsible_department': 'Samling og Forskning (KM
S)', 'items_0_current_location_name': 'Sal 228', 'items_0_frame_notes
_0': 'Bagklædning: true', 'items_0_frame_notes_1': 'Mikroklimaramme:
false', 'items_0_content_person_0': 'Hammershøi, Vilhelm', 'items_0_d
imensions_0_notes': '645 x 514 x 75 mm', 'items_0_dimensions_0_part':
'brutto', 'items_0_dimensions_0_type': 'hojde', 'items_0_dimensions_0
_unit': 'cm', 'items_0_dimensions_0_value': '64.5', 'items_0_dimensio
ns_1_notes': '645 x 514 x 75 mm', 'items_0_dimensions_1_part': 'brutt
o', 'items_0_dimensions_1_type': 'bredde', 'items_0_dimensions_1_uni
t': 'cm', 'items_0_dimensions_1_value': '51.4', 'items_0_dimensions_2
_notes': '645 x 514 x 75 mm', 'items_0_dimensions_2_part': 'brutto',
'items_0_dimensions_2_type': 'dybde', 'items_0_dimensions_2_unit': 'c
m', 'items_0_dimensions_2_value': '7.5', 'items_0_dimensions_3_note
s': '520 x 395 mm', 'items_0_dimensions_3_part': 'netto', 'items_0_di
mensions_3_type': 'hojde', 'items_0_dimensions_3_unit': 'cm', 'items_
0_dimensions_3_value': '52', 'items_0_dimensions_4_notes': '520 x 395
```

.. Hvis jeg skal arbejde med min "flatten json fil" så skal det flade data ind i en ramme, som eks. en DataFrame. Det kan gøres på forskellige niveauer.

Mit fokus i denne opgave er at kigge lidt på anskaffelsestidspunktet (de mange værker er anskaffet til SMK/dvs. tidspunktet/dato) - er der nogle mangler her, og kan det synliggøres? Kan man se, hvilke documenter som henvises til igen og igen ? Hvor mange gange har et værk været udstillet? Og hvornår..

jeg har derfor foretaget en mellem-handling for at kunne anvende json_normalize (måske en omvej, men trods alt en vej..) og jeg har arbejdet lidt med indhold i to facetter - documentation og exhibitions (begge med indeholdte dictionaries..)

In [12]:

```
#dic_flattened = [flatten_json(d) for d in json['items']]
dic_flattened = flatten_json(json_data['items'][0])

print(dic_flattened)
```

{'id': '1180006281_object', 'created': '2019-08-07T03:43:03Z', 'modifi
ed': '2019-08-12T09:23:28Z', 'acquisition_date_precision': '1971-12-3
1', 'responsible_department': 'Samling og Forskning (KMS)', 'current_l
ocation_name': 'Sal 228', 'frame_notes_0': 'Bagklædning: true', 'frame
_notes_1': 'Mikroklimaramme: false', 'content_person_0': 'Hammershøi,
Vilhelm', 'dimensions_0_notes': '645 x 514 x 75 mm', 'dimensions_0_par
t': 'brutto', 'dimensions_0_type': 'hojde', 'dimensions_0_unit': 'cm',
'dimensions_0_value': '64.5', 'dimensions_1_notes': '645 x 514 x 75 m
m', 'dimensions_1_part': 'brutto', 'dimensions_1_type': 'bredde', 'dim
ensions_1_unit': 'cm', 'dimensions_1_value': '51.4', 'dimensions_2_not
es': '645 x 514 x 75 mm', 'dimensions_2_part': 'brutto', 'dimensions_2
_type': 'dybde', 'dimensions_2_unit': 'cm', 'dimensions_2_value': '7.
5', 'dimensions_3_notes': '520 x 395 mm', 'dimensions_3_part': 'nett
o', 'dimensions_3_type': 'hojde', 'dimensions_3_unit': 'cm', 'dimensio
ns_3_value': '52', 'dimensions_4_notes': '520 x 395 mm', 'dimensions_4
_part': 'netto', 'dimensions_4_type': 'bredde', 'dimensions_4_unit':
'cm', 'dimensions_4_value': '39.5', 'documentation_0_title': 'Svend Ha
mmershøi - en kunstner og hans tid', 'documentation_0_author': 'unknow
n', 'documentation_0_notes': 'ill. p. 16', 'documentation_0_shelfmar
k': 'C 39041', 'documentation_1_title': 'Hammershøi', 'documentation_1
_author': 'unknown', 'documentation_1_notes': 'kat.nr. 11', 'documenta
tion_1_shelfmark': 'C 41297', 'documentation_2_title': 'Historier från
Danmark', 'documentation_2_author': 'unknown', 'documentation_2_note
s': 'kat. nr. 16, ill. p. 58', 'documentation_2_shelfmark': 'C 42101',
'documentation_3_title': 'Vilhelm Hammershøi: the poetry of silence',
'documentation_3_author': 'unknown', 'documentation_3_notes': 'ill. p.
43', 'documentation_3_shelfmark': 'C 42160', 'documentation_4_title':
'Vilhelm Hammershøi: Kunstneren og hans Værk', 'documentation_4_autho
r': 'Alfred Bramsen & S.Michaëlis', 'documentation_4_notes': 'kat.nr.
81', 'documentation_5_title': 'Hammershøi: Værk og liv', 'documentatio
n_5_author': 'Poul Vad', 'documentation_5_notes': 'afb. p. 68, omtalt
p.66', 'documentation_6_title': 'Vilhelm Hammershøi', 'documentation_6
_author': 'Felix Krämer', 'documentation_6_notes': 'kat. 8., afb. p. 3
4, omt. p. 149', 'documentation_6_shelfmark': 'ka2003-075', 'documenta
tion_7_title': 'Vilhelm Hammershøi: Mesterværker af Vilhelm Hammershøi
fra SMK', 'documentation_7_author': 'Cecilie Høgsbro', 'documentation_
7_notes': 'afb. p. 115', 'documentation_8_title': 'Vilhelm Hammershøi:
På sporet af det åbne billede', 'documentation_8_author': 'Annette Ros
envold Hvidt', 'documentation_8_notes': 'afb. p. 506', 'exhibitions_0_
exhibition': 'Painting Tranquility: Masterworks by Vilhelm Hammershøi
from SMK - The National Gallery of Denmark', 'exhibitions_0_date_star
t': '2015-10-16T00:00:00.000Z', 'exhibitions_0_date_end': '2016-02-26T
00:00:00.000Z', 'exhibitions_0_venue': 'The American-Scandinavian Foun
dation, New York', 'exhibitions_1_exhibition': 'Vilhelm Hammershøi: Th
e Poetry of Silence', 'exhibitions_1_date_start': '2008-06-24T00:00:0
0.000Z', 'exhibitions_1_date_end': '2008-09-07T00:00:00.000Z', 'exhibi
tions_1_venue': 'Royal Academy of Arts', 'exhibitions_2_exhibition':
'Vilhelm Hammershøi', 'exhibitions_2_date_start': '2003-03-22T00:00:0
0.000Z', 'exhibitions_2_date_end': '2003-06-29T00:00:00.000Z', 'exhibi
tions_2_venue': 'Hamburger Kunsthalle', 'exhibitions_3_exhibition': 'T
ales from Denmark', 'exhibitions_3_date_start': '2007-09-28T00:00:00.0
00Z', 'exhibitions_3_date_end': '2008-01-27T00:00:00.000Z', 'exhibitio
ns_3_venue': 'Ateneum, Helsinki', 'materials_0_material': 'lærred', 'm
aterials_1_material': 'olie', 'object_names_0_name': 'maleri', 'produc

tion_0_creator': 'Hammershøi, Vilhelm', 'production_0_creator_date_of_birth': '1864-05-15T00:00:00.000Z', 'production_0_creator_date_of_death': '1916-02-13T00:00:00.000Z', 'production_0_creator_nationality': 'dansk', 'production_0_creator_history': 'Tegninger: Td 768 Sk 19 (?)', 'production_0_creator_lref': '29700_person', 'production_date_0_start': '1890-01-01T00:00:00.000Z', 'production_date_0_end': '1890-12-31T00:00:00.000Z', 'production_date_0_period': '1890', 'techniques_0_technique': 'Olie på lærred', 'titles_0_title': 'Selvportræt', 'titles_0_language': 'da-DK', 'titles_0_type': 'MUSEUM', 'titles_1_language': 'en-US', 'titles_1_translation': 'Self-Portrait', 'number_of_parts': 1, 'object_number': 'KMS6692', 'object_url': 'https://api.smk.dk/api/v1/art/?object_number=kms6692', 'iiif_manifest': 'https://api.smk.dk/api/v1/iiif/manifest/?id=kms6692', 'production_dates_notes_0': 'Datering if. Bramsen & Michaëlis 1918 ', 'public_domain': True, 'rights': 'https://creativecommons.org/share-your-work/public-domain/cc0/', 'on_display': True, 'alternative_images_0_mime_type': 'image/tiff', 'alternative_images_0_iiif_id': 'https://iip.smk.dk/iiif/jp2/KMS6692.tif.jp2', 'alternative_images_0_iiif_info': 'https://iip.smk.dk/iiif/jp2/KMS6692.tif.jp2/info.json', 'alternative_images_0_width': 4061, 'alternative_images_0_height': 5538, 'alternative_images_0_size': 67503956, 'alternative_images_0_thumbnail': 'https://iip.smk.dk/iiif/jp2/KMS6692.tif.jp2/full/!1600,/0/default.jpg', 'alternative_images_0_native': 'https://iip.smk.dk/iiif/jp2/KMS6692.tif.jp2/full/full/0/native.jpg', 'alternative_images_0_orientation': 'portrait', 'image_mime_type': 'image/tiff', 'image_iiif_id': 'https://iip.smk.dk/iiif/jp2/KMS6692.tif.reconstructed.tif.jp2', 'image_iiif_info': 'https://iip.smk.dk/iiif/jp2/KMS6692.tif.reconstructed.tif.jp2/info.json', 'image_width': 2808, 'image_height': 3764, 'image_size': 14959624, 'image_thumbnail': 'https://iip.smk.dk/iiif/jp2/KMS6692.tif.reconstructed.tif.jp2/full/!1600,/0/default.jpg', 'image_native': 'https://iip.smk.dk/iiif/jp2/KMS6692.tif.reconstructed.tif.jp2/full/full/0/native.jpg', 'image_cropped': True, 'image_orientation': 'portrait', 'has_image': True, 'colors_0': '#555555', 'colors_1': '#555555', 'colors_2': '#000000', 'colors_3': '#5A5A5A', 'colors_4': '#9B9B9B'}

In [13]:

```
flatten_data = json_normalize(json_data)
```

In [14]:

```
json_normalize(json_data)
```

Out[14]:

	autocomplete	corrections	found	items	offset	rows
0	[Hammershøi, Vilhelm]	[]	104	[{'id': '1180006281_object', 'created': '2019-...	0	200

In [15]:

```
print(json_normalize(json_data['items']))
```

```
28                                                   NaN
29    [{'mime_type': 'image/tiff', 'iiif_id': 'https...
..                                                   ...
74    [{'mime_type': 'image/tiff', 'iiif_id': 'https...
75    [{'mime_type': 'image/tiff', 'iiif_id': 'https...
76    [{'mime_type': 'image/tiff', 'iiif_id': 'https...
77                                                   NaN
78    [{'mime_type': 'image/tiff', 'iiif_id': 'https...
79                                                   NaN
80                                                   NaN
81                                                   NaN
82                                                   NaN
83                                                   NaN
84                                                   NaN
85                                                   NaN
86    [{'mime_type': 'image/tiff', 'iiif_id': 'https...
87                                                   NaN
88                                                   NaN
89    [{'mime_type': 'image/tiff', 'iiif_id': 'https...
90    [{'mime_type': 'image/tiff', 'iiif_id': 'https...
```

In [16]:

```
for item_no in range( len(json_data['items']) ) :
    if 'documentation' not in json_data['items'][item_no] :
            json_data['items'][item_no]['documentation'] = []

    if 'exhibitions' not in  json_data['items'][item_no] :
            json_data['items'][item_no]['exhibitions'] = []
```

Øvelsen her går på at hente data udelukkende fra **documentation feltet & exhibition feltet** og dermed få en ny json_data fil at udarbejde specifikke DataFrames med:

In [17]:

```
doc_df = json_normalize(json_data['items'], record_path='documentation', meta=['id'
                        meta_prefix="item.", errors='ignore')
print(doc_df.head())
print(doc_df.dtypes)
print(doc_df.size)
print(doc_df.shape)
```

```
                         author                    notes shelfmark  \
0                       unknown               ill. p. 16   C 39041
1                       unknown               kat.nr. 11   C 41297
2                       unknown  kat. nr. 16, ill. p. 58   C 42101
3                       unknown               ill. p. 43   C 42160
4  Alfred Bramsen & S.Michaëlis               kat.nr. 81       NaN

                                        title            item.id
0   Svend Hammershøi - en kunstner og hans tid  1180006281_object
1                                 Hammershøi  1180006281_object
2                      Historier från Danmark  1180006281_object
3    Vilhelm Hammershøi: the poetry of silence  1180006281_object
4  Vilhelm Hammershøi: Kunstneren og hans Værk  1180006281_object
author       object
notes        object
shelfmark    object
title        object
item.id      object
dtype: object
2050
(410, 5)
```

In [18]:

```
exhibitions_df = json_normalize(json_data['items'], record_path='exhibitions', meta=
print(exhibitions_df.head())
print(exhibitions_df.dtypes)
print(exhibitions_df.shape)
print(exhibitions_df.size)
```

```
                  date_end                date_start  \
0  2016-02-26T00:00:00.000Z  2015-10-16T00:00:00.000Z
1  2008-09-07T00:00:00.000Z  2008-06-24T00:00:00.000Z
2  2003-06-29T00:00:00.000Z  2003-03-22T00:00:00.000Z
3  2008-01-27T00:00:00.000Z  2007-09-28T00:00:00.000Z
4  2004-11-28T00:00:00.000Z  2004-04-01T00:00:00.000Z

                                          exhibition  \
0  Painting Tranquility: Masterworks by Vilhelm H...
1          Vilhelm Hammershøi: The Poetry of Silence
2                                Vilhelm Hammershøi
3                                Tales from Denmark
4                                           Clinch!

                                       venue             item.id
0  The American-Scandinavian Foundation, New York  1180006281_object
1                          Royal Academy of Arts  1180006281_object
2                           Hamburger Kunsthalle  1180006281_object
3                              Ateneum, Helsinki  1180006281_object
4                                        Sølvgade  1180000445_object
date_end      object
date_start    object
exhibition    object
venue         object
item.id       object
dtype: object
(163, 5)
815
```

In [19]:

```
print(exhibitions_df)
```

```
20   2001-05-06T00:00:00.000Z  2001-10-07T00:00:00.000Z
21   2005-09-25T00:00:00.000Z  2005-05-21T00:00:00.000Z
22   2016-06-18T00:00:00.000Z  2016-03-02T00:00:00.000Z
23   2012-05-19T00:00:00.000Z  2012-02-03T00:00:00.000Z
24   2016-06-18T00:00:00.000Z  2016-03-02T00:00:00.000Z
25   2003-06-29T00:00:00.000Z  2003-03-22T00:00:00.000Z
26   2008-01-27T00:00:00.000Z  2007-09-28T00:00:00.000Z
27   2008-09-07T00:00:00.000Z  2008-06-24T00:00:00.000Z
28   2015-05-17T00:00:00.000Z  2015-02-05T00:00:00.000Z
29   2004-11-07T00:00:00.000Z  2004-08-21T00:00:00.000Z
..                        ...                       ...
133  2016-05-14T00:00:00.000Z  2016-02-17T00:00:00.000Z
134  2008-01-27T00:00:00.000Z  2007-09-28T00:00:00.000Z
135  2007-01-07T00:00:00.000Z  2006-09-01T00:00:00.000Z
136  2017-09-09T00:00:00.000Z  2017-04-07T00:00:00.000Z
137  2004-11-23T00:00:00.000Z  2004-10-30T00:00:00.000Z
138  2011-05-08T00:00:00.000Z  2011-02-26T00:00:00.000Z
139  2018-08-11T00:00:00.000Z  2018-04-18T00:00:00.000Z
140  1993-12-31T00:00:00.000Z  1993-12-31T00:00:00.000Z
141  2016-05-07T00:00:00.000Z  2016-02-10T00:00:00.000Z
```

In [20]:

```
def number_in_list( mylist ) :
    # Return the number of items in a list. If the value is not a list, just return
    mylen = 0
    if type(mylist) is list:
        mylen = len( mylist)
    return mylen
```

In [21]:

```
all_df['number_of_exhibitions'] = all_df.apply( lambda row:
                        number_in_list(row['exhibitions']), axis=1 )

all_df['number_of_documents']   = all_df.apply( lambda row:
                        number_in_list(row['documentation']), axis=1 )
```

Funktionen *value_counts* kan anvendes til at producere en dataframe over mine udvalgte data, inkl. optælling og procent - dokumentationen for det anvendte er fundet her: [https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.value_counts.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.value_counts.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.value_counts.html)

In [22]:

```
visualise = all_df['number_of_exhibitions'].value_counts(dropna=False,
                                                         normalize=True)
```

In [23]:

```
print(visualise * 100) # in order to get the number in percent
```

```
0     58.653846
1     10.576923
2      9.615385
5      6.730769
6      3.846154
4      2.884615
3      2.884615
14     0.961538
12     0.961538
10     0.961538
9      0.961538
7      0.961538
Name: number_of_exhibitions, dtype: float64
```

In [24]:

```
print(type(visualise))
```

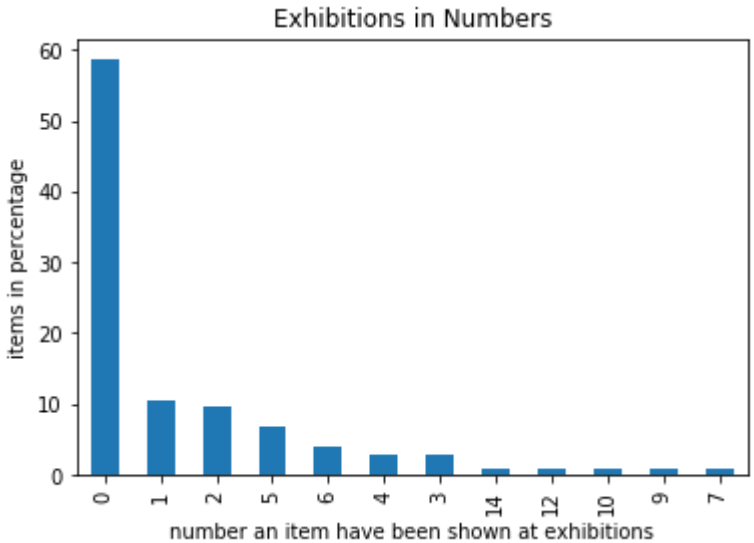
```
<class 'pandas.core.series.Series'>
```

In [25]:

```
(visualise * 100).plot.bar()
plt.title('Exhibitions in Numbers')
plt.xlabel('number an item have been shown at exhibitions')
plt.ylabel('items in percentage')
```

Out[25]:

```
Text(0, 0.5, 'items in percentage')
```



In [26]:

```
all_df['acquisition_date_precision'].head(200)
```

Out[26]:

```
0      1971-12-31
1      1916-12-31
2      1997-12-31
3      1972-12-31
4      1924-12-31
5      1928-12-31
6      1970-12-31
7      1970-12-30
8      1916-10-30
9      2005-05-13
10     2008-11-06
11     1998-09-10
12     1916-12-31
13     1924-12-31
14     1984-05-02
15     1908-12-31
16     1974-12-31
17     2012-09-17
18     2015-09-27
19     1938-12-31
20     1904-01-15
21     1988-03-24
22     1971-12-31
23     1924-12-31
24     1896-12-31
25     1916-12-31
26     1908-12-31
27     1926-02-16
28     1912-12-31
29     1970-12-31
          ...
74     1988-12-31
75     1992-09-02
76     2010-03-22
77     2016-04-18
78     1948-12-31
79     1964-12-31
80            NaN
81            NaN
82            NaN
83     2006-08-23
84     1948-12-31
85            NaN
86     1971-12-31
87            NaN
88     1998-10-01
89     2011-11-04
90     1918-12-31
91     1975-06-10
92     1955-04-26
93            NaN
94            NaN
95     1986-12-31
```

```
96            NaN
97     1914-12-31
98            NaN
99     1915-03-27
100    1899-12-31
101    1932-12-31
102    1955-05-04
103    1981-12-30
Name: acquisition_date_precision, Length: 104, dtype: object
```

In [27]:

```
regex = r'^(\d{4})'
all_df['acquisition_year'] = all_df['acquisition_date_precision'].str.extract(r'^(\d
all_df.loc[25:, 'acquisition_year'].head(10)
```

Out[27]:

```
25    1916
26    1908
27    1926
28    1912
29    1970
30    1977
31     NaN
32    1971
33     NaN
34    1934
Name: acquisition_year, dtype: object
```

---------- over and out ----------