Tina Hegelund Hyldahl Martin - HSR607
Mikkel Birk Nielsen - FXK415
Kasper Senika Larsen - PHM536
Christian Runge Hansen - QKT122

Københavns Universitet
Open Data Science
Informationsvidenskab
Vinter 19/20

Tina Hegelund Hyldahl Martin - HSR607
Mikkel Birk Nielsen - FXK415
Kasper Senika Larsen - PHM536
Christian Runge Hansen - QKT122

Københavns Universitet
Open Data Science
Informationsvidenskab
Vinter 19/20

Portfolio 1

Opgave 1.

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

Ved at se på filen i Notepad, har vi kunne se, at de forskellige informationer er listet i rækkefølge, pænt struktureret og adskilt af kommaer, der fungerer som kolonner. Der er 8 kolonner: Om passageren overlevede (angivet boolsk, i 0 eller 1 - vi antager at 1 betyder personen overlevede), passagerklasse, navn, køn, alder, antal søskende/ægtefæller om bord, antal forældre/børn om bord, samt hvor meget de har betalt for overfarten.

Det er værd at nævne, at gifte kvinders navne er sat i parentes, mens mandens fulde navn også står ved personen.

b. Mangler der data?

Ja, på Titanic var der ca. 1.300 passagerer. Disse passagerer var fordelt med 325 på 1. klasse, 285 på 2. klasse og 706 på 3. klasse, plus ca. 900 besætningsmedlemmer. Dette datasæt indeholder kun 887 passagerer. Derfor mangler der ca. 500 passagerer i dette regnestykke, foruden besætningen.

Derudover har vi diskuteret hvilke typer af data, der også kunne være angivet for at skabe et mere fyldestgørende datasæt. Der kunne godt have været flere brugbare informationer om passagerne i form af: nationalitet, anvendt type af valuta til betaling, billet nr., kabine nr. og hvilken havn, der blev anvendt til påstigning.

Samtidig savner vi en bedre separation mellem søskende, ægtefæller, forældre og børn i kolonnerne.

Derudover kunne data omkring passagernes erhverv og religion også være interessante at undersøge.

Opgave 2.

```
# (følgende opgave 2-5 er Python kode, derfor udeladt at fjerne #)
import pandas as pd
titanic = pd.read_csv("titanic.csv", sep=',')
# print(titanic)
print(titanic.head()) # Viser de første 5 linier af filen for at få en forståelse af dataen
print(len(titanic)) # Antal rows
print(titanic.shape) # Antal rows & columns
print(titanic.size) # Antal celler i hele filen
print(titanic.columns) # Navnene til columns
print(titanic.dtypes) # Data typer af filen int, str eller float.
# Kilde: https://datacarpentry.org/python-socialsci/08-Pandas/index.html
```

Opgave 3.

```
# Antal af personer der overlevede = Summen af Survived column
print('Antal af personer der overlevede:', sum(titanic.Survived))
# Gennemsnitsalderen på passagererne = mean af Age column
print('Gennemsnitsalderen for alle passagerer på Titanic:', titanic['Age'].mean())
# Median alder på passagerne = median af Age column
print('Median alderen på alle passagerer på Titanic:', titanic['Age'].median())
# Mindste værdi altså yngste passager = min af Age column
print('Alder på yngste passager:', titanic['Age'].min())
# Max værdi ældste passager = max af Age colum
print('Alder på ældste passager:', titanic['Age'].max())
# Antal af passagerer på hver klasse = value_count af Pclass
print(titanic['Pclass'].value_counts())
# Gennemsnitspris pr billet. Dog er billet ikke angivet i valuta = mean af Fare column
print('Gennemsnitspris pr billet alle passagerer på Titanic:', titanic['Fare'].mean())
# Kilde: https://datacarpentry.org/python-socialsci/10-aggregations/index.html
```

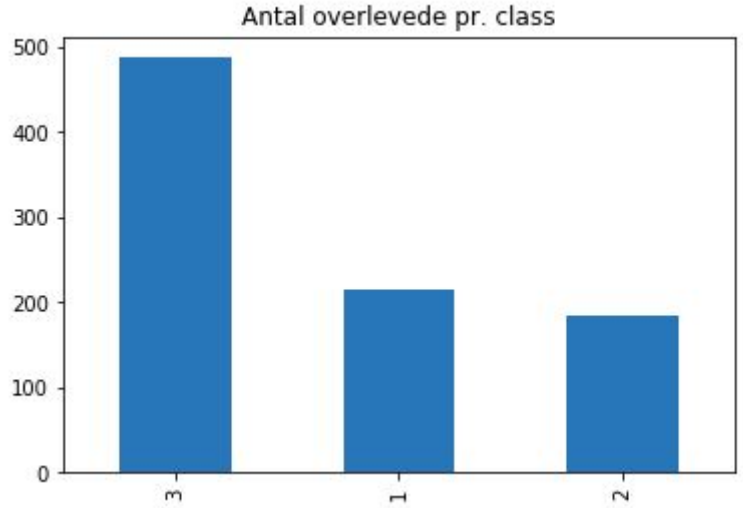
Opgave 4.

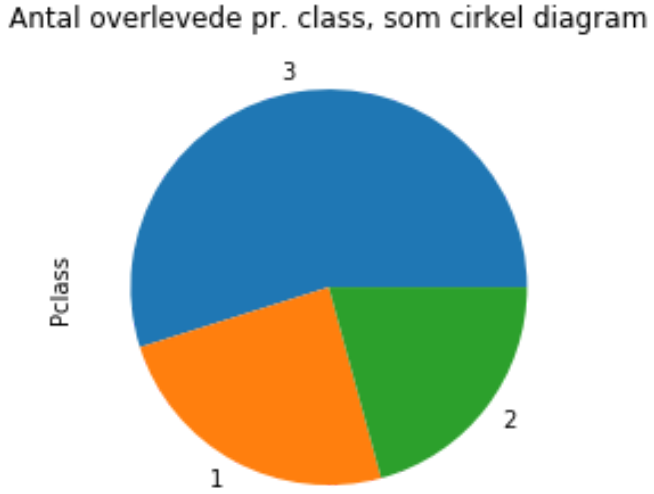
```
#Hvor mange har samme efternavn
new = titanic['Name'].str.split(r'((\w+$))', n = 0, expand = True)
new[1].value_counts()
new = titanic['Name'].str.rsplit(pat = r'((\w+^))', expand = True)
# Column name split str op, r = reverse \w er helt ord og $ er anker
new = (titanic['Name'].str.split().str[-1])
# -1 betyder at der startes med den sidste karakter i string.
print(new.value_counts())
# https://docs.python.org/3/library/stdtypes.html#str.rsplit
```

Opgave 5.

```
# Pivot-tabel over hvor mange passagerer på hver klasse & hvilken klasse havde fleste omkomne?
pd.pivot_table(titanic, values=['Survived'], columns='Pclass', aggfunc='count')
# Aggregeret funktion med optælling.
titanic.groupby(['Pclass', 'Survived'])['Survived'].count()
# Gruppering opdelt i Pclass og Survived
df = titanic[['Pclass','Survived']]
print(df.shape)
# Antal personer på inddelt i hver passagerklasse
print(pd.pivot_table(df, values='Survived', columns='Pclass', aggfunc='count'))
# Antal overlevende inddelt i hver passagerklasse.
```

```
print(pd.pivot_table(df, values='Survived', columns='Pclass', aggfunc='sum'))
# Kilde:
https://pandas.pydata.org/pandas-docs/stable/user_guide/groupby.html#named-aggregation

from nltk.probability import FreqDist
fdist = FreqDist(new.value_counts())
fdist.most_common(10)

# Vi kan visualisere vores data i python blandt andet gennem et søjlediagram og et cirkeldiagram, som ses nedenfor.

# Vi kan ændre, hvilken form vi ønsker at vores visualisering tager ved at ændre kind. For at få et søjlediagram har vi sat kind = bar og for at få et cirkeldiagram har vi sat kind = pie.

# Søjlediagram over antal overlevede pr. klasse
barchart = df['Pclass'].value_counts().plot(kind='bar', title ="Antal overlevede pr. class")

# Cirkeldiagram over antal overlevede pr. klasse
piechart = df['Pclass'].value_counts().plot(kind='pie', title ="Antal overlevede pr. class, som cirkeldiagram")
```


Antal overlevede pr. class
500
400
300
200
100
0
3
1
2


Antal overlevede pr. class, som cirkel diagram
3
Pclass
2
1

Litteraturliste portfolio 1

Datacarpentry.org (2019) python-social sci, 08-pandas, set online 9 december 2019
https://datacarpentry.org/python-socialsci/08-Pandas/index.html

Datacarpentry.org (2019) python-social sci, 10-aggregations, set online 9 december 2019
https://datacarpentry.org/python-socialsci/10-aggregations/index.html

Pandas.pydata.org (2019) Pandas docs: user guide, group by, set online 9 december 2019
https://pandas.pydata.org/pandas-docs/stable/user_guide/groupby.html#named-aggregation

Python.org (2019) The Python Standard Library, set online 9 december 2019
https://docs.python.org/3/library/stdtypes.html#str.rsplit

Portfolio 2

For this assignment we first have our python code followed by our key findings and reflections.

Python code:

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

# This creates two subdirectories called "theguardian" and "collection"
ARTICLES_DIR = join('theguardian', 'collection')
makedirs(ARTICLES_DIR, exist_ok=True)

# 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

# Change this for your API key:
MY_API_KEY = 'f820d1dc-b679-45fb-aa15-443d1c44924f'

API_ENDPOINT = 'http://content.guardianapis.com/search'
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
}

# day iteration from here:
# http://stackoverflow.com/questions/7274267/print-all-day-dates-between-two-dates
```

```
# Update these dates to suit your own needs.
start_date = date(2019, 4, 1)
end_date = date(2019,10, 31)

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):
        # then let's download it
        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'])
            # if there is more than one page
            current_page += 1
            total_pages = data['response']['pages']

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

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

Assignment 1. The collection

Import TheGuardian OpenApi

```
import json
import os
from nltk.corpus import stopwords as sw

directory_name = "theguardian/collection/"

ids = list()
texts = list()
sections = 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:
                id = article['id']
                fields = article['fields']
                text = fields['bodyText'] if fields['bodyText'] else ""
                ids.append(id)
                texts.append(text)
                section = article['sectionId']      # Id name each article gets by The Guardian
                sections.append(section) # Adding each item to a list as above "sections = list()"

print("Number of ids: %d" % len(ids))
print("Number of texts: %d" % len(texts))
```

Assignment 2. Pre-process and describe your collection

```
sect = set(sections) # Changing the list into a set, meaning that no duplicates of the section titles will appear. It thereby creates a list where each unique name appears only once.
# print(sect) # This could print the whole list of unique categories from the data set.
len(sect) # Counts the list of categories.

# Unique count of each of the ID names with the count of each category. Showing each how many articles there are under the ID name
import numpy as np
unique, counts = np.unique(sections, return_counts=True)
dict(zip(unique, counts))

# How many characters there are combined, through all the data.
all_lengths = list()
for text in texts:
    all_lengths.append(len(text))
```

```
print("Total sum of characters in dataset: %i" % sum(all_lengths))

# When performing a tokenization we can split up the strings and thereby count the total number of words. This method is without any tokenization tools.
word_count = 0
for text in texts:
   words = text.split()
   word_count = word_count + len(words)
word_count

# To get the unique words we do a word split but now also extend the words.
all_words = list()
for text in texts:
  words = text.split()
  all_words.extend(words)
unique_words = set(all_words)
unique_word_count = len(unique_words)
print("Unique word count: %i" % unique_word_count)

# The average word length is found by first finding all the individual word lengths, and then calculating the average from that.
total_word_length = 0
for word in all_words:
   total_word_length = total_word_length + len(word)
average_word_length = total_word_length / len(all_words)
print("Average word length: %.6f" % average_word_length)

# To find out how many sentences there are in total, we first select the end of sentence marker for where a sentence should finish.
def end_of_sentence_marker(character):
   if character in ['.', '?', '!']: # In our case we made '.', '?', '!' our sentence splitters.
     return True
   else:
     return False

def split_sentences(texts):
   sentences = []
   start = 0
   for end, character in enumerate(text): # The enumerate adds a counter to an iterable and returns it in a form of enumerate object, that then can be used in loops.
       if end_of_sentence_marker(character):
          sentence = text[start: end + 1]
          sentences.append(sentence)
          start = end + 1
```

```
    return sentences

all_sentences = list()
for text in texts:
  sentences = split_sentences(text)
  all_sentences.extend(sentences)
sentence_count = len(all_sentences)
sentence_count

# The average number of words in a sentence
words_per_sentence = word_count / sentence_count
print("words per sentence: %.6f" % words_per_sentence)

# The following code shows a random sample of the unique words. This is to give an idea of what the
words looks like. Here we see that some words are not typical words e.g. numbers '£22.99' or
abbreviations like 'JCRA' or that the words have punctuation marks around them and therefore
appear as a different word than the same word without them: look" vs. look
import random
random.sample(unique_words, 20)

from nltk.tokenize import word_tokenize

# Word tokenization of the documents and converting of big letters to small ones.
tokens = list()
for text in texts:
  tokens_in_text = word_tokenize(text)
  for token in tokens_in_text:
    if token.isalpha():
      tokens.append(token.lower())

# Creating a new list from the token list with stopwords removed
stopWords = set(sw.words('english'))

swwords = list()
for w in tokens:
    if w not in stopWords:
        swwords.append(w)

print("Str.split word count: %i" % word_count) # Wordcount using str.split
print("Word count: %i" % len(tokens)) # Wordcount using nltk tokenizer
print("Word count with stopwords removed: %i" % len(swwords)) # Wordcount with stopwords
removed using nltk tokenizer
unique_tokens = set(tokens)
unique_swwords = set(swwords)
```

```
print("Str.split unique word count: %i" % unique_word_count) # Unique wordcount using str.split
print("Unique word count: %i" % len(unique_tokens)) # Unique wordcount using nltk tokenizer
print("Unique word count with stopwords removed: %i" % len(unique_swwords)) # Unique
wordcount with stopwords removed using nltk tokenizer

import random
random.sample(unique_swwords, 20)

# Here we make a subset of the entire dataset. 2 lists are being made to include both the section
id´s: “sport” & “football” and the indexes for the original list.
idxes = []
subtexts = []
for i, section in enumerate(sections):
   if section in ['sport','football']:
      idxes.append(i)
      subtexts.append(texts[i])
len(idxes) # Test to see how many files there are under the sections “sport” & “football”
respectively. These numbers can be compared to the list of all sections, where the total number of
articles per section can be observed.
```

Assignment 3. Select articles using a query

```
# To create the document-term matrix for the dataset we need to use CountVectorizer,
# which can be imported with sklearn. To transform our data we must also use
model_vect.fit_transform.
# When running to code, it tells us that there are 2.395.815 elements in our document matrix
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import stopwords as sw
model_vect = CountVectorizer(stop_words= stopWords, token_pattern=r'[a-zA-Z\-][a-zA-Z\-]{2,}')
data_vect = model_vect.fit_transform(subtexts)
print('Shape: (%i, %i)' % data_vect.shape) # This shows how many documents 6808 x how many
terms 87615 there are in our subset. We changed the initial dataset from only containing articles
from starting on the 1st of September to ending on the 30th of October, to starting on the 1st of
April to ending on the 30th of October. We did so because we wanted to show our results on a larger
datascale.
data_vect

# In this line of code we demonstrate how we're able to find the index placement of the 10 most
used
# words in our documents. When using. A1 we're able to present results in an array, which
# can be described as a line of numbers. The [:10] tells that we want it as a top 10. We can change
this
# number if we want to show more or fewer results eg. 20 or 5.
counts = data_vect.sum(axis=0).A1
```

```
top_idxs = (-counts).argsort()[:10]
top_idxs

# Here we use inverted_vocabulary to assign the actual words belonging to the top-10 indexes in the sub-sets.
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 in subset: %s" % top_words)

# Frequency/amount of times the words are being used in the entire dataset. So we can compare it to the most used word in the subsets.
from nltk.probability import FreqDist
fdist = FreqDist(swwords)
fdist.most_common(30)

# This line is not important in itself, but it creates a submatrix that will be used later on.
import random
some_row_idxs = random.sample(range(0,len(subtexts)), 10)
print("Selection: (%s x %s)" % (some_row_idxs, top_idxs))
sub_matrix = data_vect[some_row_idxs, :][:, top_idxs].todense()
sub_matrix

# The code here transforms the tf-idf model.
from sklearn.feature_extraction.text import TfidfTransformer
model_tfidf = TfidfTransformer()
data_tfidf = model_tfidf.fit_transform(data_vect)

# The freqs function helps us sort the top 10 words.
freqs = data_tfidf.mean(axis=0).A1
top_idxs = (-freqs).argsort()[:10].tolist()
top_words = [inverted_vocabulary[idx] for idx in top_idxs]
(top_idxs, top_words)

# Now we can use the submatrix and the transformed tf-idf, to create a scheme of the top 10 most used words and their weight in 10 random documents.
import pandas as pd
sub_matrix = data_tfidf[some_row_idxs, :][:,top_idxs].todense()
df = pd.DataFrame(columns=top_words, index=some_row_idxs, data=sub_matrix)
df

# Query terms that we think we will find in our subsets.
terms = ['tottenham', 'hotspurs', 'hotspur', 'spurs', 'stadium', 'new', 'home']
terms
```

```
# The count of each term in the subsets, how many times it is used.
term_idxs = [model_vect.vocabulary_.get(term) for term in terms]
term_counts = [counts[idx] for idx in term_idxs]
term_counts

# Calculate the term weights
idfs = model_tfidf.idf_
term_idfs = [idfs[idx] for idx in term_idxs]
term_idfs

# Creates a matrix for the query words and their weight in the chosen subsets
dfi = pd.DataFrame(columns=['count', 'idf'], index=terms, data=zip(term_counts,term_idfs))
dfi
```

Assignment 4. Model and visualize the topics in your subset

```
# 4 components
from sklearn.decomposition import LatentDirichletAllocation
model_lda = LatentDirichletAllocation(n_components=6, random_state=0)
data_lda = model_lda.fit_transform(data_vect)
np.shape(data_lda)

# Here we show the top 20 words and their individual weight related to each of the 4 components
for i, term_weights in enumerate(model_lda.components_):
    top_idxs = (-term_weights).argsort()[:20]
    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 small matrix to illustrate which document is the most likely to match one of the topics that was
found in the code above.
topic_names = ["Topic" + str(i) for i in range(model_lda.n_components)]
doc_names = ["Doc" + str(i) for i in range(len(subtexts))]
df = pd.DataFrame(data=np.round(data_lda, 2), columns=topic_names, index=doc_names).head(10)
df.style.applymap(lambda val: "background: red" if val>.3 else '', )

# In this code we're able to take a random document and show its index placement, topic vector,
which of the 4 components it relates the most to. Lastly we're shown the entire document text
doc_idx = random.randint(0,len(subtexts)-1)
print('Doc idx: %d' % doc_idx)
topics = data_lda[doc_idx]
print('Topic vector: %s' % topics)
vote = np.argsort(-topics)[0]
```

```
print('Topic vote: %i' % vote)
subtexts[doc_idx]

# Word Cloud
from wordcloud import WordCloud
import matplotlib.pyplot as plt

for i, term_weights in enumerate(model_lda.components_):
   top_idxs = (-term_weights).argsort()[:20]
   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=200,height=200,
max_words=20).generate_from_frequencies(word_freqs)
   plt.subplot(2, 3, i+1)
   plt.imshow(wc)

# The following code compares each of the topics from earlier to ALL texts in our subset and shows
which document is most connected to each topic.
df = pd.DataFrame(data=data_lda, columns=topic_names)
df['class'] = idxes
df = df.groupby('class').mean().round(2)
df.style.applymap(lambda val: "background: red" if val>.5 else '', )

# These lines make unique words from our subsets, where stopwords, small letters, and tokenizer
are used. This has been done to get plot frequency distribution and Zipf's law graph.
subtokens = list()
for subtext in subtexts:
  tokens_in_subtext = word_tokenize(subtext)
  for subtoken in tokens_in_subtext:
    if subtoken.isalpha():
      subtokens.append(subtoken.lower())
stopWords = set(sw.words('english'))
subwords = list()
for w in subtokens:
   if w not in stopWords:
      subwords.append(w)

# Plot frequency distribution
from nltk.probability import FreqDist
fdist = FreqDist(subwords)
fdist.most_common(30)
import matplotlib.pyplot as plt
fdist.plot(30,cumulative=False)
plt.show()
```

```
# Zipf's law
ranks = range(1, len(fdist) + 1)
freqs = list(fdist.values())
freqs.sort(reverse = True)
plt.plot(ranks, freqs, '-')
plt.xscale('log')
plt.yscale('log')
plt.show()

# Here we try to answer our research question: With which topics was the football club Tottenham mentioned?
# We have the previously selected and used terms from assignment 3. First, we combine the 7 query words.
query = " ".join(terms)
query

# Then changing the query in to a sparse matrix with 1 row.
query_vect_counts = model_vect.transform([query])
query_vect = model_tfidf.transform(query_vect_counts)
query_vect

# We then compare all terms with our research question, where the percentage shows how well they fit with our query.
from sklearn.metrics.pairwise import cosine_similarity
sims = cosine_similarity(query_vect, data_tfidf)
sims

# Here we sort the documents by how well they fit our query. The most likely is at the top.
sims_sorted_idx = (-sims).argsort()
sims_sorted_idx

# The document that fits our query best is shown here and you get the whole article. The article is: https://www.theguardian.com/football/blog/2019/apr/03/tottenham-new-stadium-spurs-numbers-game-crystal-palace which is about the new stadium that Tottenham hotspurs got in March.
subtexts[sims_sorted_idx[0,0]]

# Second query using only "tottenham" and "spurs" to find articles only related to these terms.
TotSputerms = ['tottenham', 'spurs']

# In these lines we make a vectorizer model of how many times tottenham and spurs are mentioned (frequency), and what their idf is. The two numbers together lets us calculate the weight of the terms.
totterm_idxs = [model_vect.vocabulary_.get(term) for term in TotSputerms]
```

```
totterm_counts = [counts[idx] for idx in totterm_idxs]
Totidfs = model_tfidf.idf_
Totterm_idfs = [Totidfs[idx] for idx in totterm_idxs]
totdf = pd.DataFrame(columns=['count', 'idf'], index=TotSputerms,
data=zip(totterm_counts,Totterm_idfs))
totdf

# We join the terms tottenham and spurs into a string, then we transform them into a model
vectorizer. We compare the similarity between the query and each document.
tsquery = " ".join(TotSputerms)
tsquery_vect_counts = model_vect.transform([tsquery])
tsquery_vect = model_tfidf.transform(tsquery_vect_counts)
from sklearn.metrics.pairwise import cosine_similarity
tssims = cosine_similarity(tsquery_vect, data_tfidf)
tssims_sorted_idx = (-tssims).argsort()

# In this section we show the top 5 most related articles to our query.
print("First article most fitting query: (%s)" % (subtexts[tssims_sorted_idx[0,0]]))
print("Second article most fitting query: (%s)" % (subtexts[tssims_sorted_idx[0,1]]))
print("Third article most fitting query: (%s)" % (subtexts[tssims_sorted_idx[0,2]]))
print("Fourth article most fitting query: (%s)" % (subtexts[tssims_sorted_idx[0,3]]))
print("Fifth article most fitting query: (%s)" % (subtexts[tssims_sorted_idx[0,4]]))

# Mini matrix showing cosine similarity between articles and the research question. This shows
which article is the most likely to have the words from our query to answer the research question.
print("Shape of 2-D array similarity from query: (%i, %i)" % (len(tssims), len(tssims[0,:])) )
tsdf = pd.DataFrame(data=zip(tssims_sorted_idx[0,:], tssims[0,tssims_sorted_idx[0,:]]),
columns=["Article Id", "Similarity in %"])
tsdf[0:10]
```

Key Findings and Reflections

For this assignment we decided to work with TheGuardian OpenApi. This decision was based on the fact that TheGuardian OpenApi was more similar to the 20newsgroup dataset, we had worked with in class.
It also seemed to be presented more orderly than the transcripts of danish news broadcasts, which would give more options for processing the data. With the lack of punctuation in the transcripts of danish news broadcasts, processing would be difficult, as we would not be able to define end of sentence markers.
Furthermore TheGuardian OpenApi would give a wider range of topics to research, which would give more freedom in choosing a research question.

When exploring the dataset on The Guardian Open Platform (https://open-platform.theguardian.com/explore/), we noticed that all articles were listed by multiple different categories:

▼ 0: {} 11 keys
id: "football/2019/oct/07/football-quiz-winning-streaks-liverpool"
type: "article"
sectionId: "football"
sectionName: "Football"
webPublicationDate: "2019-10-07T10:37:20Z"
webTitle: "Football quiz: winning streaks"
webUrl: "https://www.theguardian.com/football/2019/oct/07/football-quiz-winning-streaks-liverpool"
apiUrl: "https://content.guardianapis.com/football/2019/oct/07/football-quiz-winning-streaks-liverpool"
isHosted: false
pillarId: "pillar/sport"
pillarName: "Sport"

From these categories we chose to work with “id” and “sectionId” when importing the dataset.
We chose to work with “sectionId” over the very similar “sectionName” as it had the name of the section listed in small letters, which is easier to work with in python, as python differentiates between capital letters and small letters.
We chose to work with the sections to be able to do a broad categorization of all the topics for further analysis.

The overview provided by The Guardian Open Platform helped us understand what the different categories were used to show, as well as where they would show when viewing the articles in a browser:

Data processing

During the pre-processing phase we noticed that the total word counts using the str.split and the word count tokenizer did not show a big difference when compared to each other. We assume that this is because the data in The Guardian collection is well-edited news materiel.

We used random samples of the unique words we got when using str.split. This was done to get an idea of what the unique words look like. Here we noticed that some words were not typical words e.g. numbers: '£22.99' or abbreviations: 'JCRA' or that the words had punctuation marks around them and therefore appeared as a different word than the same word without them: look" vs. look. Therefore we used the nltk tokenizer to weed out all of these unique words that were not actual words. The nltk tokenizer automatically converts all capital letters to small letters, to make sure all unique words only appear once.
Compared to our previous work with the 20newsgroup dataset we also noticed that among our sampled unique words there were fewer spelling mistakes. We assume that this is due to the fact, that the data in The Guardian collection is published news materiel, that is edited before publication.

It is interesting to notice, that when removing the stopwords from the full dataset with 30 million words, almost half of all the words are removed. But when working with the unique word count, applying the stopwords only removes 147 unique words.

When looking into the most used words for the complete dataset, all the top words: 'said', 'one', 'would', 'people', 'also', 'new', 'time', 'like', 'first', 'says' - are very common words that are likely to appear in the majority of all articles.

After this count we decided to add more stopwords: 'said', 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten', 'would', 'could', 'says', 'also'. This gave a new list of the most used words for the complete dataset: 'people', 'new', 'time', 'us', 'like', 'first', 'last', 'years', 'government', 'back'.
We also did the list of top ten most used words for the “Football” and “Sport” sections twice, before and after adding more stopwords.
Some of the top words from the list of most used words for the complete dataset, were also among the most used words for the “Football” and “Sport” sections. The words on the list of most used words for these two sections, that were not on the list of most used words for the complete dataset, give a good indication that they are related to football and sports: 'one', 'first', 'game', 'back', 'two', 'time', 'last', 'england', 'ball', 'said'.
Words like 'game', 'england', and 'ball' are commonly used to describe sports and football. The fact that 'england' is among the top words also gives a clue that England’s own national teams are in focus.
In the second run of top ten most used words for the “Football” and “Sport” sections with added stopwords, it is even more clear that the words are related to sport: 'first', 'game', 'back', 'time', 'last', 'england', 'ball', 'team', 'world', 'min'. Here the words 'team', 'world', and 'min' are added to the list.

Research question and restrictions

We have decided to focus on the “Football” and “Sports” sections. We have chosen these sections because they are both larger sections with many articles, but with comparable and similar topics.

We have decided to use “With which topics was Tottenham (football team) mentioned?” as our research question, as it was recommended to use a named entity in order to not overly restrict the number of topics in our subset. It is a quite specific named entity as we felt like anything less specific like e.g. “ball” or “football” would be too non-specific to look for within the “Football” and “Sports” sections.

Before making a final decision, we also experimented with different queries such as: “With which topics was Brexit discussed?” But we are all tired of hearing about Brexit. We also talked about how interesting it was, that Brexit is a part of the politics section, rather than it being in its own separate section, considering how this topic dominates the politics sections of many news providers at the moment.

We also discussed how looking into the musician “A$AP Rocky” could be difficult, as his name might be caught in the nltk filter due to the use of $ in his name.

In the following of our research question: With which topics was Tottenham (football team) mentioned? We expect to find the topics: Tottenham, hotspurs, spurs, stadium, hotspur, new, and home. This is because we know that the players on the Tottenham football team are referred to as the “hotspurs” or just the “spurs”, and that they had a new stadium built in the spring.
We have chosen our query due to the expectation of these terms being used by the journalists; e.g. we expect the nickname of the football team to be used to refer to the team.

Furthermore, we expect to find topics such as: NFL, football, new stadium, transfers, (Christian) Eriksen, Champions League, and National Team.

With the top 5 most relevant articles that we found through our query, we try to answer our research question: With which topics was Tottenham (football team) mentioned?

1
https://www.theguardian.com/football/2019/jul/02/tanguy-ndombele-tottenham-medical-lyon-roma-toby-alderweireld

This article is about the transfer of Tanguy Ndombele to Tottenham, whilst also discussing other possible incoming and outgoing transfers.

2
https://www.theguardian.com/football/blog/2019/jun/02/mauricio-pochettino-harry-kane-tottenham-champions-league

This article is focused on the manager of Tottenham and reflects over future choices following their champions league final defeat.

3
https://www.theguardian.com/football/blog/2019/may/29/why-i-gave-away-my-champions-league-final-ticket-to-watch-it-with-my-dad

This article is about how a man trades away one expensive ticket for the final match of champions league, so he can watch it with his dad instead.

4
https://www.theguardian.com/football/live/2019/sep/08/chelsea-v-tottenham-womens-super-league-live

It is interesting to notice that this article is about women's football - still the Tottenham spurs - but not the men's team like we expected.

5
https://www.theguardian.com/football/2019/apr/30/tottenham-hotspur-ajax-champions-league-semi-final-first-leg-match-report

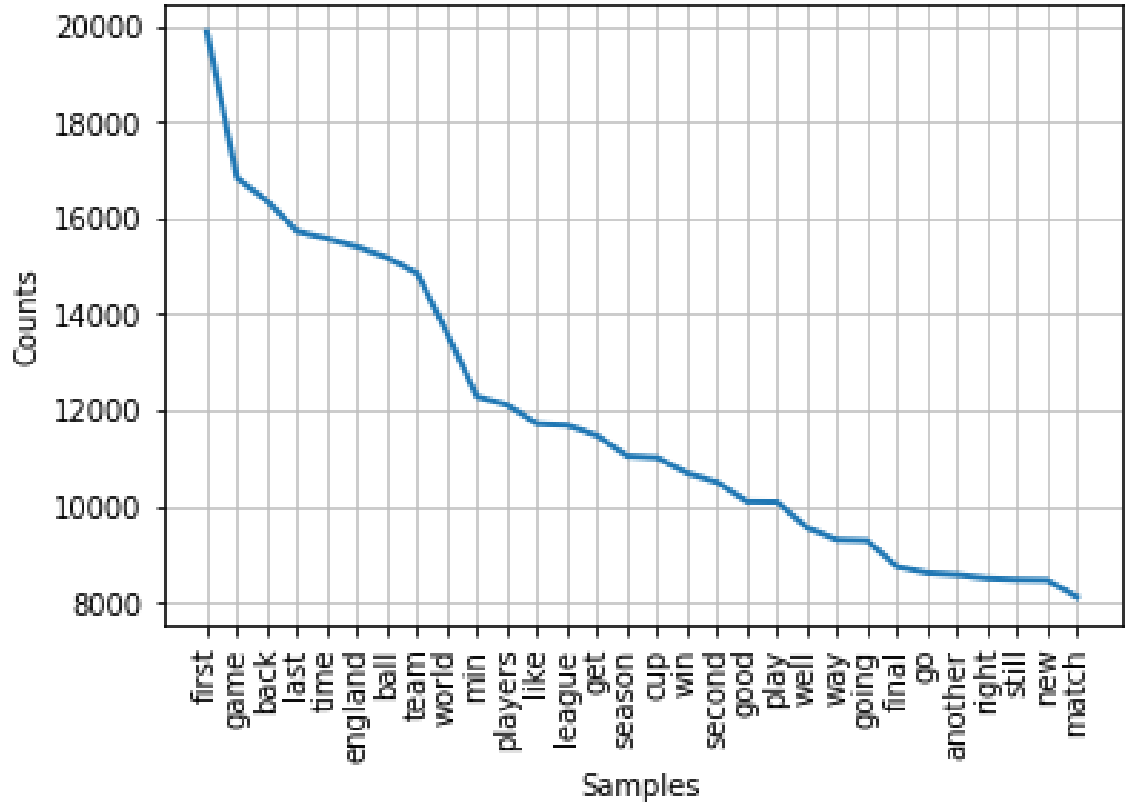
This article is a match summary following a champions league match between Tottenham and Ajax.

Model and visualization of topics

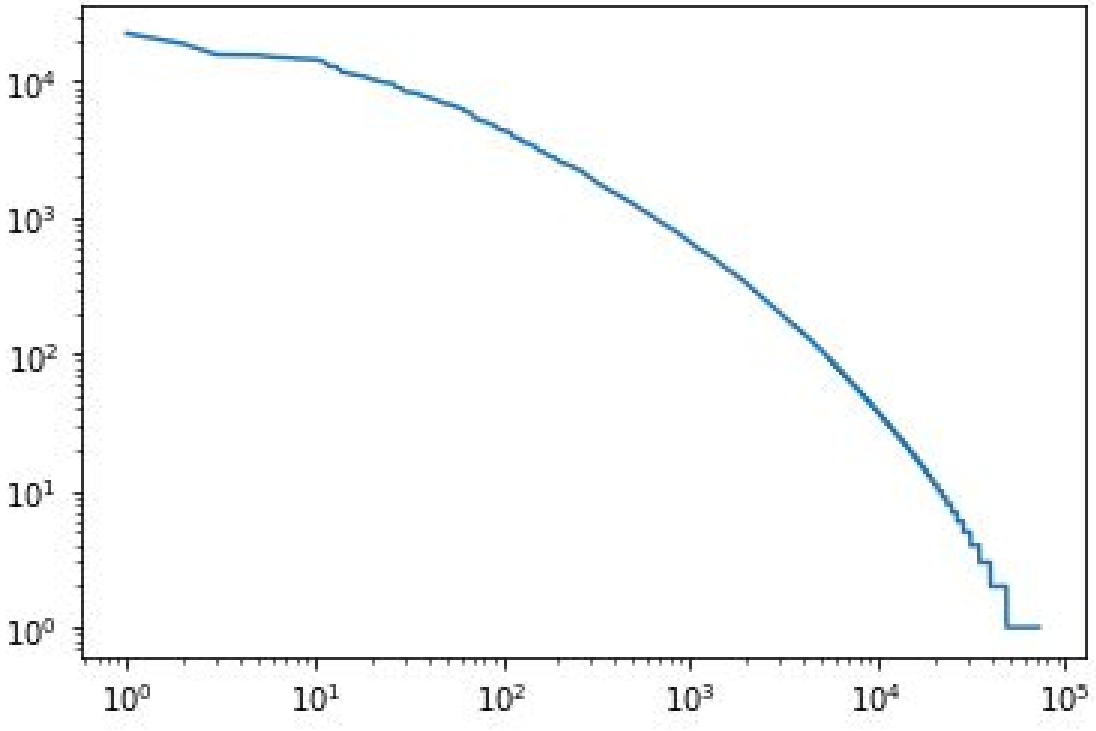
Query terms presented in a tf-idf matrix:

query term	count	idf
tottenham	1975	4.775314
hotspurs	2	10.600331
hotspur	312	6.328305
spurs	2310	5.153593
stadium	2795	4.323687
new	61406	1.614886
home	24605	2.249586

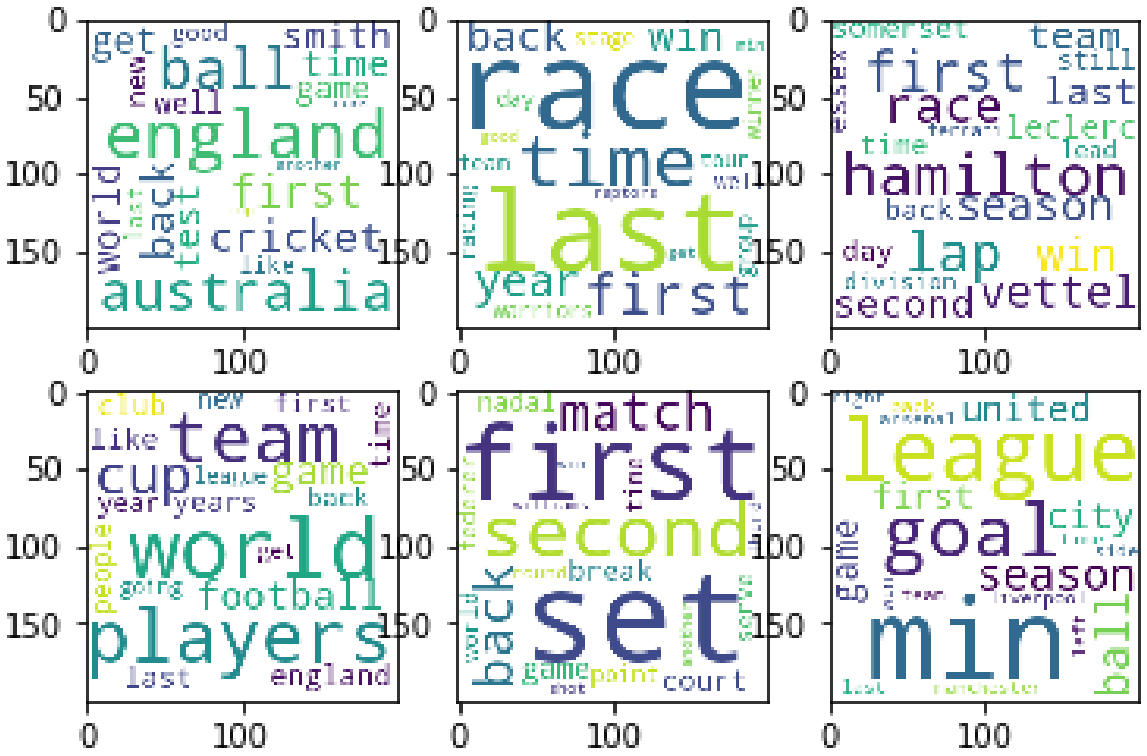
Frequency distribution of the top 30 most used words in the football & sports subset:



Zipf's law graph - showing that in this dataset the frequency of words is inversely proportional to its place in the table:



WordClouds for the topics with the highest weight from 6 random documents:



Topic 0:
In the WordCloud for the first topic, among the words with the highest weight we find “england” and “australia”. Based on that we can conclude that there’s talk about the national teams and not the leagues. The word with the sixth highest weight is “cricket” and that sums up which sport there is talked about and with the aforementioned national teams, what they are playing. Another thing to notice is that “test” and “game” have a very similar weights, that can indicate that this is a test game between England and Australia, but nothing that can be confirmed for sure.

Topic 1:
Two of the most indicating words in topic 1 is “warriors” and “raptors”. These are both basketball teams in the NBA, and therefore they could indicate that the article is about basketball. There are also some words there are kind of hard to connect with basketball, such as “race”, “tour” and “stage”. Therefore it is hard to say if it is basketball the article is about but other words such as “win” and “min” could also lead against its a basketball article.

Topic 2:
The topics for this article are clearly indicating what type of sport the article is about. The word with the highest weight is “hamilton” and two other words with a respectively lower weight is “vettel” and “leclerc”. These are names of Formula 1 drivers, therefore this article is likely about Formula 1. Three other words with a higher weight is “race”, “somerset”, and “essex”, which tells us the location of the race.

Topic 3:
In this WordCloud the three words with the highest weights are “world”, “players”, and “team”. These words can be used within a large number of sports. However, by looking further down the list of words, we see words like “football” and “england” are placed as 6th and 7th highest weighing words. This allows us to conclude that this article is about football and probably about the World Cup in football since cup is also one of the most mentioned words.

Topic 4:
In this WordCloud the highest weighing word is “set”, which obviously leads one to think that the article is about tennis. There are more words that back this up: “court”, “nadal”, “serve”, “federer”, and “williams”. Three of these words are the names of prominent tennis players, while the words “court” and “serve” are used often within tennis.

Topic 5:
Lastly, the final WordCloud has names of famous football clubs as some of the highest weighing words; “city”, “united”, “liverpool”, and “arsenal”. The highest weighing word; “min” leads us to believe that this article is in fact a live blog post, that discusses some of the top football clubs in England during a game day eg. “at min 5 liverpool are close to take the lead against City”.

Bibliography Portfolio 2

Downey, A. (2015). *Think Python. How to think like a computer scientist.* 2. Ed, Version 2.4.0. Needham, Mass.: Green Tea Press. Retrieved August 21, 2019 from http://greenteapress.com/thinkpython2/thinkpython2.pdf

TheGuardian.com (2019). *Open Platform*, set online 9 december, 2019 https://open-platform.theguardian.com/explore/

Ignatow, G. & Mihalcea, R. (2018). *An Introduction to Text Mining : Research Design, Data Collection, and Analysis*. Los Angeles, CA.: Sage.

Lavin, M.J. (2019). Analyzing Documents with TF-IDF. *The Programming Historian* 8

Sweigart, A. (2015). *Automate the boring stuff with Python : practical programming for total beginners* (6th printing.). San Francisco: No Starch Press.

Portfolio 3

Opgave 1)

Beskriv (med referencer) hvorfor institutionen har en API tilgang til deres datasæt, hvad det er blevet brugt til og/eller hvad de tænker det kunne bruges til?

Når en kulturinstitution som SMK vælger at gøre deres samling tilgængelig via API, skyldes det et ønske om en øget digital tilstedeværelse. Denne digitalisering kommer til udtryk gennem projektet; "SMK Open". Om projektet skriver SMK selv; "Hensigten med SMK Open er at åbne for SMK's værksamling og sætte en væsentlig del af danskernes kulturarv fri" (U/Å). Når SMK, som de selv skriver, sætter danskernes kulturarv fri, er det et forsøg på, at give alle adgang til værkerne uden omkostninger - denne adgang gives gennem et API. Dermed ikke sagt, at det kun er åbne data som kan tilgås gennem et API. F.eks. den data vi har brugt i vores portfolio 2 fra The Guardian, er ikke 100% åbne data, idet man skal anmode om en API nøgle, og denne anmodning kan af forskellige årsager blive afvist. Tilgængelighed handler ikke udelukkende om at værkerne skal kunnes opleves digitalt, de skal også kunne bruges i undervisningssammenhænge. Peter Leth (2014) skriver i bogen; Sharing is caring, at "Tilgængelighed handler ikke kun om, at et værk kan ses på en hjemmeside" og "Læring er en aktiv proces, hvor eleven skaber forståelse og vigtig kommunikation ved at producere udtryk. Derfor er det vigtigt, at viden må kopieres, gengives i besvarelser, bearbejdes, remixes og bruges på nye måder" (s. 251). Citatet fra Peter Leth er relevant for SMK's vedkommende, fordi de i rapporten beskriver, at blandt andet skolelærere er vigtige samarbejdspartnere i arbejdet med digitaliseringen af institutionens værksamling. Det giver sig selv, at man ikke kan få lov til at arbejde med et originalt værk - f.eks. af P.S. Krøyer, da værket let kan blive beskadiget.

SMK's udbytte af API-tilgangen kan opsummeres som værende et værktøj der muliggør, at alle har adgang til deres samling, samt et udgangspunkt for undervisning.

Som et konkret eksempel på, hvordan SMK åbner op for deres samling, kan vi kigge mod HACK 4 DK, hvor museet inviterer alle interesserede til at hacke deres samling og på kreativ vis udvikle nye måder og idéer til, hvordan samling kan bruges (HACK4DK 2019 U/Å). Dette er også et eksempel på, hvordan SMK bruger deres samling i en undervisningssammenhæng - ikke undervisning i den kontekst, som vi kender fra universitetet eller den undervisning, som Leth omtaler. Men der er et klart læringsudbyttet ved HACK 4 DK for både deltagerne og SMK selv.

Opgave 2)

Sammensæt en URL der søger efter kulturinstitutionens elementer (fx. genstande, kunstværker, personer, avisartikler, osv.) i APIen (gerne 100+ stk) med output i JSON. Load denne URL i en JSON beautifier og beskriv indholdet:

URL: https://api.smk.dk/api/v1/person/search/?keys=pe*er*&offset=0&rows=100

Vores URL tager udgangspunkt i en søgning på “pe*er*”, hvor trunkeringstegnet er indsat midt i og i slutningen af ordet. Vores oprindelige tanke var at søge på navnet "Peter". Ved at bruge trunkering i vores søgning kan vi få resultater, hvor kunstnere med navne som; Peter, Peder, Petersen, Pedersen, Perrier og Peeters optræder. Samtidig har vi fået et resultat på en kunstner, David Jensen, som er med i vores søgning, fordi stedet for hans død er Sankt Petersburg. Trunkeringstegnet tillader at et eller flere bogstaver kan indgå på tegnets position og erstatte det. Ofte ville man bruge tegnet “?” til at maskere et bogstav midt i søgetermen, når man anvender boolske operatører, men vi fandt frem til at bruge "*" gav os det ønskede resultat. Når vi indsætter vores URL i en JSON beautifier, får vi 70 items. Disse 70 items har hver især yderligere metadata informationer under sig.

2) a) Hvilken typer metadata er der for hver element?

For hvert element er der et id, dato for oprettelse og dato for modificering. Derudover er der angivet værker (works), fødselsdato og fødselssted (birth_date_end, birth_date_prec, birth_date_start, birth_place, birth_date_end), dødsdato og dødssted (death_date_end, death_date_prec, death_date_start, death_place), fornavn (forename), køn (gender), navn (name), navnetype (name_type), nationalitet (nationality), efternavn (surname), samt om der er et billede (has_image).

2) b) Sammenlign metadata med Dublin Core. Er der overensstemmelse?

Da Dublin Core er en universel standardisering for metadata der kan anvendes, bruges deres metadata element-sæt i rigtig mange databaser. Nogle databaser vil have brug for andre eller flere/færre metadata, alt efter hvad de skal bruges til. Derfor har SMK også flere punkter der stemmer overens med DC såsom: skaber, dato, filformat, sprog, udgiver og titel. Dublin Core er som udgangspunkt lavet til bogudgivelser, som en standardisering af udgivelserne, således at alt data om en udgivelse blev medtaget til videre brug. Dublin Core kan bruges i mange tilfælde, da det ville få brugerne til at huske vigtige elementer af data, som skal medbringes over et værk/materiale. Især ved oprettelse af en ny database vil Dublin Core være en god guideline at gennemgå, dog med en kritisk tilgang til dette.

I Dublin Core element-sættet findes følgende 15 kerneelementer af metadata. Sammenligner vi Dublin Core med metadataene fra vores datasæt, kan vi se, at følgende metadata fra er repræsenteret begge steder.

Contributor - Nej
Coverage - Nej - nærmeste der kommer vil være en livsperiode
Creator - Ja værkets skaber
Date - Dette svarer til de dato- og tidsinformationer, som er angivet ud fra kunstnerne
Description - Nej
Format - Nej
Identifier - Ja bestående i form af id på kunstnerne
Language - Nej, men nationalitet af kunstner relaterer til emnet
Publisher - Nej
Relation - Nej, det eneste ville være værker af sammen kunstner relaterer til hinanden
Rights - Nej

Source - Nej
Subject - Person/object
Title - Dette svarer til værkernes navngivning i works kolonnen. Herefter kan vi se navngivningen på alle de mange forskellige værker under hver kunstner.
Type - Dette svarer til name_types fra vores datasæt, hvor vi får bekræftet, at personen er en kunstner.

2) c) Er der nogle tal der kan bruges til statistiske beregninger (fx. årstal)?

Ja, vi kan ud fra at dette konkludere, at det er muligt at lave statistik over den metadata der eksisterer omkring elementerne, men da der er metadata der er misvisende, kan det være upræcist at lave statistik over dette. f.eks. er metadata under kunstnernes fødselsdatoer og dødsdatoer ikke indført i datasættet. Om den danske billedhugger Christian Carl Peters ved vi at han er født 26. jul. 1822 og død, 17. sep. 1899 (http://denstoredanske.dk/Dansk_Biografisk_Leksikon/Kunst_og_kultur/Billedkunst/Billedhugger/C.C._Peters). I metadataene fra SMK står der blot årstal og fødselsdatoerne er sat til start: 1. jan. med slut 31. dec. skulle man konkludere på dette, så havde C.C. Peters ifølge APIet fødselsdag 365 dage om året. Dette gør sig gældende for flere af kunstnerne, baseret ud fra en stikprøve, og det vil derfor ikke være muligt at lave præcise statistikker over dette.

2) d) Hvilke slags spørgsmål kunne besvares gennem analyse af dette datasæt? (gerne med eksempler)

Hvor mange kunstnere findes der i SMK's database, hvor navnet pe*er* indgår? Hvor er flest kunstnere født? Hvor er flest kunstnere døde? Hvilket år en/flere kunstner(e) var mest produktiv? (via graf) Hvilken kunstner har været mest produktiv? Hvem levede længst? Hvilket år blev flest af SMK's værker produceret?

Opgave 3)

Ved hjælp af Pandas og request, importer nu jeres valgte datasæt.

3 a) Beskriv og rens datasættet.

```
import requests
import pandas as pd
from pandas.io.json import json_normalize

smk_search_url = 'https://api.smk.dk/api/v1/person/search/?keys=pe*er*&offset=0&rows=100'

params = {

        'q': 'pe*er*',

        'items': 'items',
```

```
        'encoding': 'json',

        'n': 100

}

response = requests.get(smk_search_url, params=params)
print(response)
json = response.json()
df = json_normalize(json['items'])
df.head()
```

3) a) i) Hvilke data og datatyper er der?

```
df.dtypes
```

```
birth_date_end      object
birth_date_prec     object
birth_date_start    object
birth_place         object
created             object
death_date_end      object
death_date_prec     object
death_date_start    object
death_place         object
forename            object
gender              object
has_image           bool
id                  object
modified            object
name                object
name_type           object
nationality         object
surname             object
works               object
dtype: object
```

Ved første øjekast undrede det os at birth_date_start er angivet som 'object' og ikke 'integer', men da vi kiggede nærmere på kolonnen, kunne vi se at der indgår bogstaverne; T & Z, og derfor er kolonnen angivet som 'object'. Vi har i vores videre rensning af dataen forsøgt at fjerne T & Z for efterfølgende at kunne ændre kolonnen fra 'object' til 'integer'. Dette var desværre ikke muligt. Desværre er selve SMK's datastruktur er svær at arbejde med. Sammenlignet med vores arbejde med i Trove's database, virker det som om, at der ikke er tænkt over hvordan andre aktører udefra

skal kunne arbejde med de tilgængelige data gennem SMK's API. Sammenlignet med Trove's database har SMK ingen nemme visualiseringer tilgængelige.

3) a) ii) Hvor mange kolonner og rækker?

df.shape

Henholdsvis 70 kolonner og 19 rækker.

3) a) iii) Er der kolonner I ikke skal bruge?

```
to_drop = ['birth_date_end',
       'birth_date_prec',
       'birth_date_start',
        'death_date_end',
        'death_date_prec',
        'death_date_start',
        'has_image',
        'name',
        'created',
        'modified',
        'name_type',]
df.drop(to_drop, inplace=True, axis=1)
df.head()
```

Ja, vi har valgt at fjerne 11 elementer. Da flere af fødselsdatoerne og dødsdatoerne er upræcist angivet, har vi valgt at fjerne dem. Vi ville gerne have brugt årstallene til statistik, men fandt at måden de var skrevet på i SMK's database meget svære at arbejde med, da der ikke var overensstemmelse i måden de er angivet. Årstallet er skrevet i ISO8601 format, som indeholder årstal, måned, dag, time, minutter og sekunder. Tiden er omgivet af T og Z for at angive tiden og Z for at angive international tidszone, hvis tidszone er kendt skifter man Z ud med +2 alt efter hvor mange tidszoner der er plus eller minus i forhold til Greenwich Time. Dette format er oftest brugt til at lave timestamps. Vi har fjernet "has_image" da det er irrelevant for vores udvalgte statistik, om der er et billede af kunstneren i SMK's database. Vi har fjernet kolonnen "name" da vi har valgt at arbejde med de separate kolonner for henholdsvis fornavne og efternavne. Derudover har vi fjernet "created" og "modified", da vi ikke behov for at vide hvornår SMK har oprettet og modificeret data for at kunne lave statistik over dem. Desuden var alle poster, som vi testede ved en stikprøve, oprettet den samme dag, hvilket ville give meget kedelig statistik. Kolonnen "name_type" er brugt til at angive at personerne er kunstnere. Her står der det samme ud for alle poster, derfor har vi også valgt at fjerne denne.

Opgave 4)

Lav nu udtræk og beregn på det data I har udvalgt. Hvilke informationer kan det give os om de udvalgte elementer?

 # Ændring af udvalgte kolonners navne

```
df.rename(columns={'birth_date_start':'date of birth', 'birth_place':'place of birth',
'death_date_start':'date of death', 'death_place':'place of death', 'has_image':'image available',
'name':'full name'}, inplace=True)
df.head()
```

Vi har nu renset vores data. Vi har fjernet kolonner, vi ikke bruger. Derudover har vi omdøbt overskrifterne på flere af kolonnerne. Med det datasæt vi nu står med, kan man kigge på ting som; hvor mange af hvert køn er repræsenteret? Hvilke for- og efternavne er repræsenteret, samt hvilke navne er de mest hyppigt forekommende? Hvilke nationaliteter er repræsenteret og efterfølgende hvor mange af hver? Hvilke byer kunstnerne er henholdsvis født i og død i, og hvor mange gange er de byer repræsenteret i datasættet?

Disse spørgsmål bliver alle besvaret og visualiseret i opgaverne 5 og 6.

Opgave 5)

Brug value_counts() (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.value_counts.html) funktionen til at producere en dataframe over det udvalgte data, optælling og procent.

Vi har valgt at vise resultater fra opgave 5 og 6 sammen, inddelt efter tema herunder.

Da SMK's API bliver opdateret løbende, kan der forekomme ændringer i resultaterne. Gennem vores arbejde med API'et har vi oplevet ændring i resultater. Resultaterne i denne opgaver er senest opateret pr. 2/12-2019.

Opgave 6)

Visualiser det udvalgte data med procent i en graf

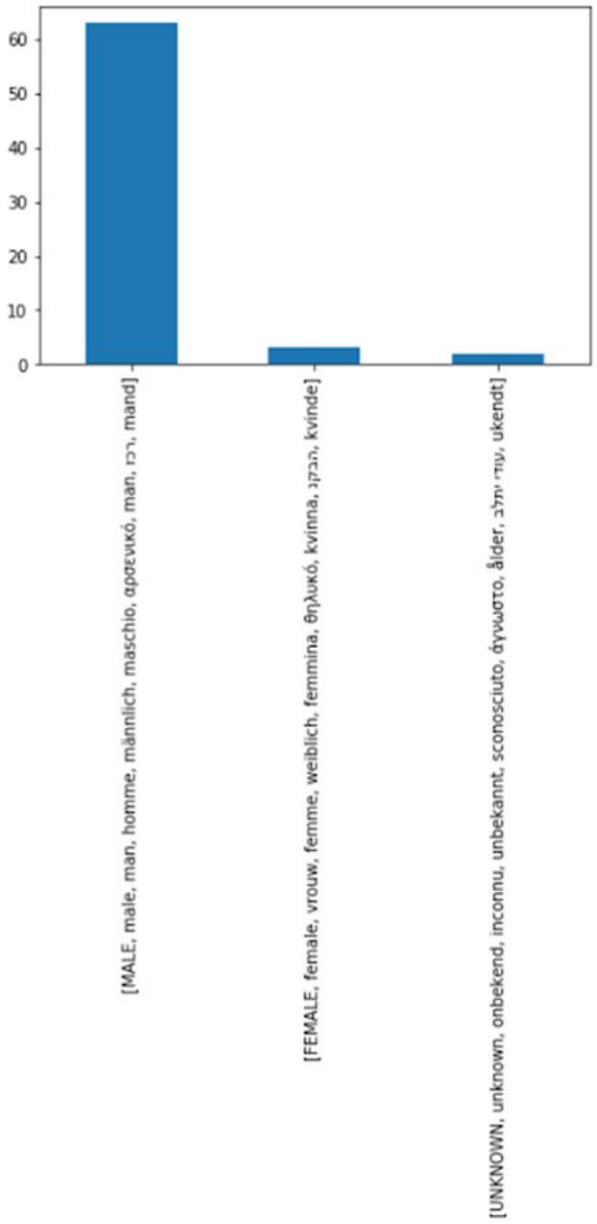
For at visualisere vores data i de forskellige temaer, har vi valgt at lave både søjle- og cirkeldiagrammer til hvert tema. Dette var både for at øve, hvordan de forskellige typer af diagrammer laves, men også en god øvelse i at finde ud af, hvilken type af diagram, der bedst egner sig til at visualisere forskellige former for data. Vi har valgt at beholde begge typer af diagrammer til alle temaer, men vi er klar over, at til f.eks. navnes fordeling, så virker cirkeldiagrammer overvældende pga. de mange afsnit, der dårligt kan læses. Ved at anvende to måder til at visualisere de samme data, kan vi og de, der læser med, vurdere for hvert enkelt tema, om det fungerer bedst med søjle- eller cirkeldiagram.

```
# Kønsfordeling
gendertal = df['gender'].value_counts()
genderprocent = df['gender'].value_counts(normalize=True).mul(100).round(1).astype(str) + '%'
```

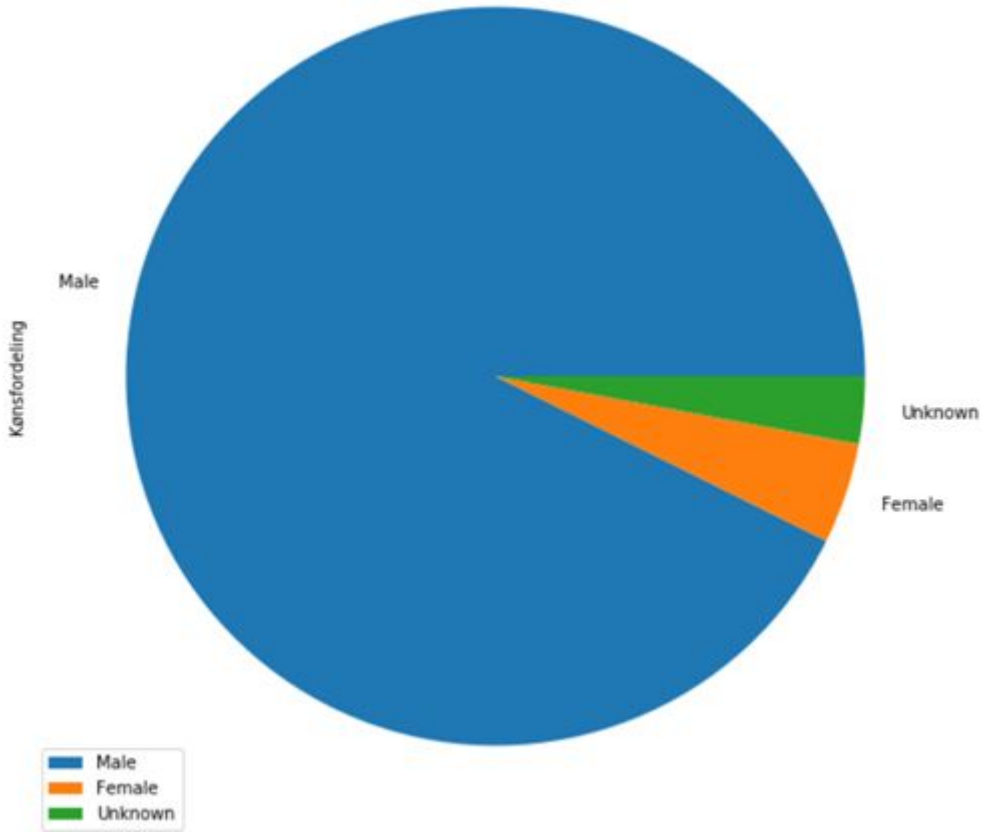
```
gender = pd.DataFrame(columns=['Antal', 'Procent'], index=['Male', 'Female', 'Unknown'],
data=zip(gendertal,genderprocent))
print(gender)
```

	Antal	Procent
Male	63	92.6%
Female	3	4.4%
Unknown	2	2.9%

```
genderbar = df['gender'].value_counts()[:70].plot(kind='bar')
# 'bar' giver en vandret akse, 'barh' giver en horizontal akse
# [:70] styrer antallet af rækker der medtages
```




```
df = pd.DataFrame({'Kønsfordeling': [63, 3 , 2],
                   'radius': [genderprocent]},
                  index=['Male', 'Female', 'Unknown'])
plot = df.plot.pie(y='Kønsfordeling', figsize=(10, 10))
```



Den store procentforskydning i køn mellem male kontra female / unknown kan forklares ved at kigge på vores url. Her søger vi efter primært mandsdominerede navne som Peter og Peder.

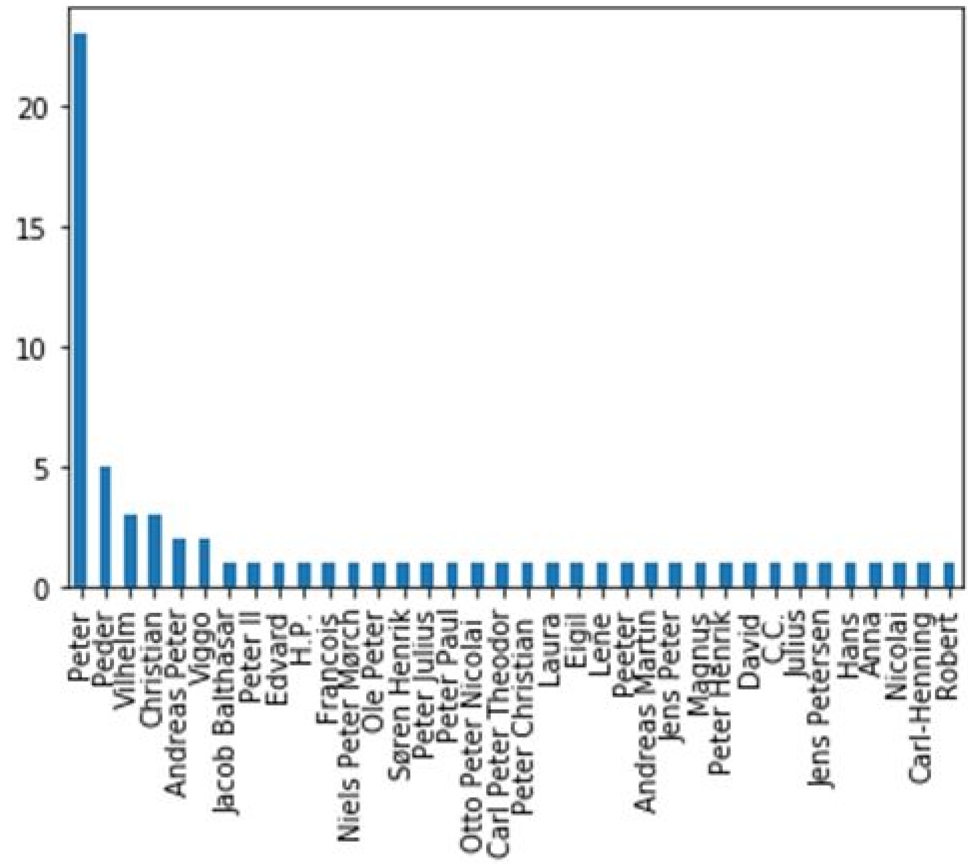
 # Fordeling af fornavne

```
fornavntal = df['forename'].value_counts()

fornavnprocent = df['forename'].value_counts(normalize=True).mul(100).round(1).astype(str) + '%'

fornavn = pd.DataFrame(columns=['Antal', 'Procent'], index=['Peter', 'Peder', 'Christian', 'Vilhelm', 'Viggo', 'Andreas Peter', 'Edvard', 'Otto Peter Nicolai', 'H.P.', 'Jens Petersen', 'Peter Christian', 'Niels Peter Mørch', 'Laura', 'Andreas Martin', 'Jens Peter', 'Julius', 'Eigil', 'Søren Henrik', 'Carl Peter Theodor', 'Magnus', 'Francois', 'Peter Julius', 'Hans', 'Peter Paul', 'Lene', 'Jacob Balthasar', 'Ole Peter', 'David', 'Peeter', 'Nicolai', 'Peter Henrik', 'Peter II', 'C.C.', 'Anna', 'Robert', 'Carl-Henning'], data=zip(fornavntal,fornavnprocent))

print(fornavn)
```

	Antal	Procent
Peter	23	33.8%
Peder	5	7.4%
Christian	3	4.4%
Vilhelm	3	4.4%

Viggo	2	2.9%
Andreas Peter	2	2.9%
Edvard	1	1.5%
Otto Peter Nicolai	1	1.5%
H.P.	1	1.5%
Jens Petersen	1	1.5%
Peter Christian	1	1.5%
Niels Peter Mørch	1	1.5%
Laura	1	1.5%
Andreas Martin	1	1.5%
Jens Peter	1	1.5%
Julius	1	1.5%
Eigil	1	1.5%
Søren Henrik	1	1.5%
Carl Peter Theodor	1	1.5%
Magnus	1	1.5%
Francois	1	1.5%
Peter Julius	1	1.5%
Hans	1	1.5%
Peter Paul	1	1.5%
Lene	1	1.5%
Jacob Balthasar	1	1.5%
Ole Peter	1	1.5%
David	1	1.5%
Peeter	1	1.5%
Nicolai	1	1.5%
Peter Henrik	1	1.5%
Peter II	1	1.5%
C.C.	1	1.5%
Anna	1	1.5%
Robert	1	1.5%
Carl-Henning	1	1.5%

```
fornavnebar = df['forename'].value_counts()[:70].plot(kind='bar')
```

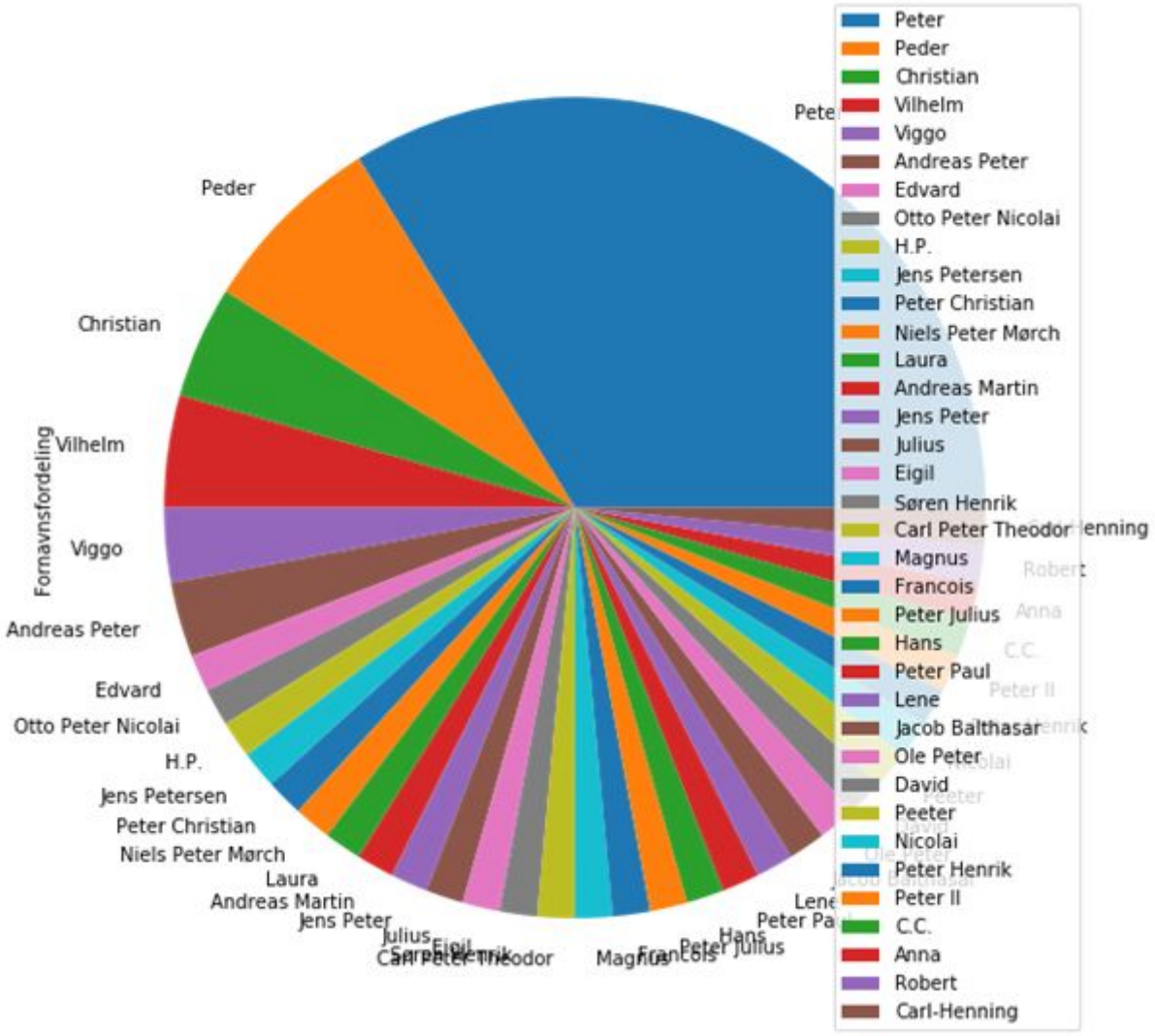


fornavndf = pd.DataFrame({'Fornavnsfordeling': [23, 5, 3, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],

'radius': [fornavnprocent]},

index=['Peter', 'Peder', 'Christian', 'Vilhelm', 'Viggo', 'Andreas Peter', 'Edvard', 'Otto Peter Nicolai', 'H.P.', 'Jens Petersen', 'Peter Christian', 'Niels Peter Mørch', 'Laura', 'Andreas Martin', 'Jens Peter', 'Julius', 'Eigil', 'Søren Henrik', 'Carl Peter Theodor', 'Magnus', 'Francois', 'Peter Julius', 'Hans', 'Peter Paul', 'Lene', 'Jacob Balthasar', 'Ole Peter', 'David', 'Peeter', 'Nicolai', 'Peter Henrik', 'Peter II', 'C.C.', 'Anna', 'Robert', 'Carl-Henning'])

pobplot = fornavndf.plot.pie(y='Fornavnsfordeling', figsize=(10, 10))



 # Fordeling af efternavne

surnametal = df['surname'].value_counts()

surnameprocent = df['surname'].value_counts(normalize=**True**).mul(100).round(1).astype(str) + '%'

surname = pd.DataFrame(columns=['Antal', 'Procent'], index=['Petersen', 'Pedersen', 'Lund', 'Peters', 'Colonia', 'Isaacz', 'Bjerke Petersen', 'Mandrup', 'Casteels', 'Sevel', 'Mønsted', 'Schøler', 'Peeters', 'Andersen', 'Candid', 'Carlsen', 'Brandes', 'Raadsig', 'Kongstad Petersen', 'Neeffs d.Æ.', 'Feilberg', 'Ilsted', 'Larsen', 'Friis', 'Hougaard', 'Rothweiler', 'Weis', 'Tyndall', 'Storm Petersen', 'Meyer Petersen', 'Land', 'Ølsted', 'Adler Petersen', 'Bonde', 'Balke', 'Perrier', 'Louis-Jensen', 'Neuchs', 'Hansen', 'Madsen', 'Jensen', 'Christensen', 'Gemzøe', 'Magnus-Petersen', 'Haas', 'Zacho', 'Bonnén', 'Als',

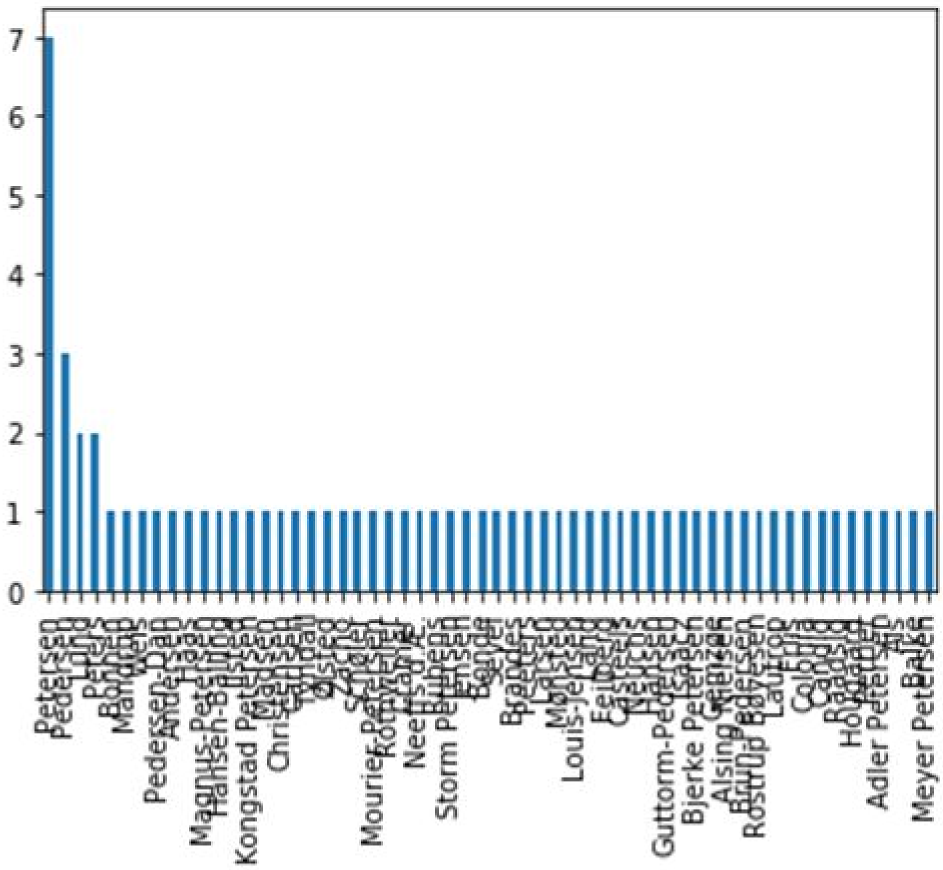
'Lautrop', 'Cramer', 'Alsing Nielsen', 'Rubens', 'Mourier-Petersen', 'Brun-Pedersen', 'Rostrup Bøyesen', 'Guttorm-Pedersen', 'Hansen-Balling', 'Pedersen-Dan'],
data=zip(surnametal,surnameprocent))

print(surname)

	Antal	Procent
Petersen	7	10.3%
Pedersen	3	4.4%
Lund	2	2.9%
Peters	2	2.9%
Colonia	1	1.5%
Isaacz	1	1.5%
Bjerke Petersen	1	1.5%
Mandrup	1	1.5%
Casteels	1	1.5%
Sevel	1	1.5%
Mønsted	1	1.5%
Schøler	1	1.5%
Peeters	1	1.5%
Andersen	1	1.5%
Candid	1	1.5%
Carlsen	1	1.5%
Brandes	1	1.5%
Raadsig	1	1.5%
Kongstad Petersen	1	1.5%
Neeffs d.Æ.	1	1.5%
Feilberg	1	1.5%
Ilsted	1	1.5%
Larsen 1	1.5%	
Friis	1	1.5%
Hougaard	1	1.5%
Rothweiler	1	1.5%
Weis	1	1.5%
Tyndall	1	1.5%
Storm Petersen	1	1.5%
Meyer Petersen	1	1.5%
Land	1	1.5%
Ølsted	1	1.5%
Adler Petersen	1	1.5%
Bonde	1	1.5%
Balke	1	1.5%
Perrier	1	1.5%
Louis-Jensen	1	1.5%

Neuchs	1	1.5%
Hansen	1	1.5%
Madsen	1	1.5%
Jensen	1	1.5%
Christensen	1	1.5%
Gemzøe	1	1.5%
Magnus-Petersen	1	1.5%
Haas	1	1.5%
Zacho	1	1.5%
Bonnén	1	1.5%
Als	1	1.5%
Lautrop	1	1.5%
Cramer	1	1.5%
Alsing Nielsen	1	1.5%
Rubens	1	1.5%
Mourier-Petersen	1	1.5%
Brun-Pedersen	1	1.5%
Rostrup Bøyesen	1	1.5%
Guttorm-Pedersen	1	1.5%
Hansen-Balling	1	1.5%
Pedersen-Dan	1	1.5%

```
surnamebar = df['surname'].value_counts()[:70].plot(kind='bar')
```

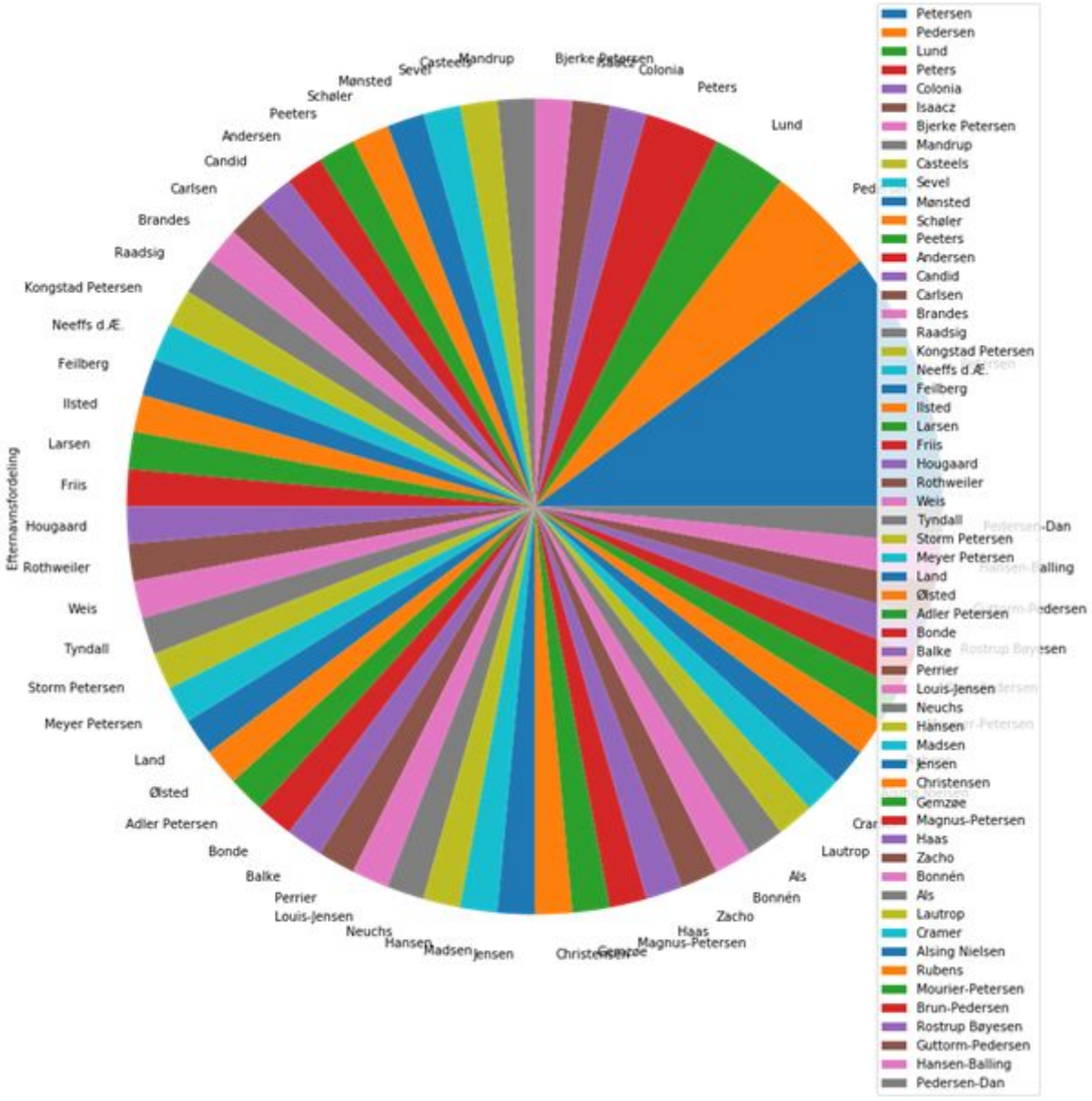


surnamedf = pd.DataFrame({'Efternavnsfordeling': [7, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],

'radius': [surnameprocent]},

index=['Petersen', 'Pedersen', 'Lund', 'Peters', 'Colonia', 'Isaacz', 'Bjerke Petersen', 'Mandrup', 'Casteels', 'Sevel', 'Mønsted', 'Schøler', 'Peeters', 'Andersen', 'Candid', 'Carlsen', 'Brandes', 'Raadsig', 'Kongstad Petersen', 'Neeffs d.Æ.', 'Feilberg', 'Ilsted', 'Larsen', 'Friis', 'Hougaard', 'Rothweiler', 'Weis', 'Tyndall', 'Storm Petersen', 'Meyer Petersen', 'Land', 'Ølsted', 'Adler Petersen', 'Bonde', 'Balke', 'Perrier', 'Louis-Jensen', 'Neuchs', 'Hansen', 'Madsen', 'Jensen', 'Christensen', 'Gemzøe', 'Magnus-Petersen', 'Haas', 'Zacho', 'Bonnén', 'Als', 'Lautrop', 'Cramer', 'Alsing Nielsen', 'Rubens', 'Mourier-Petersen', 'Brun-Pedersen', 'Rostrup Bøyesen', 'Guttorm-Pedersen', 'Hansen-Balling', 'Pedersen-Dan'])

surnameplot = surnamedf.plot.pie(y='Efternavnsfordeling', figsize=(15, 15))

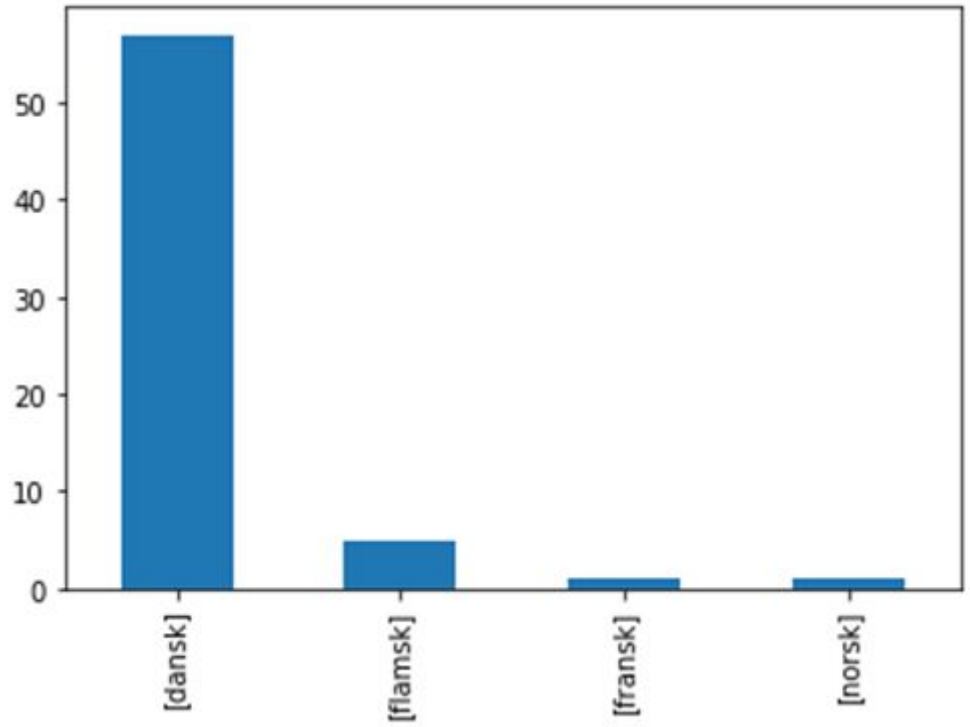


 # Fordeling af nationaliteter

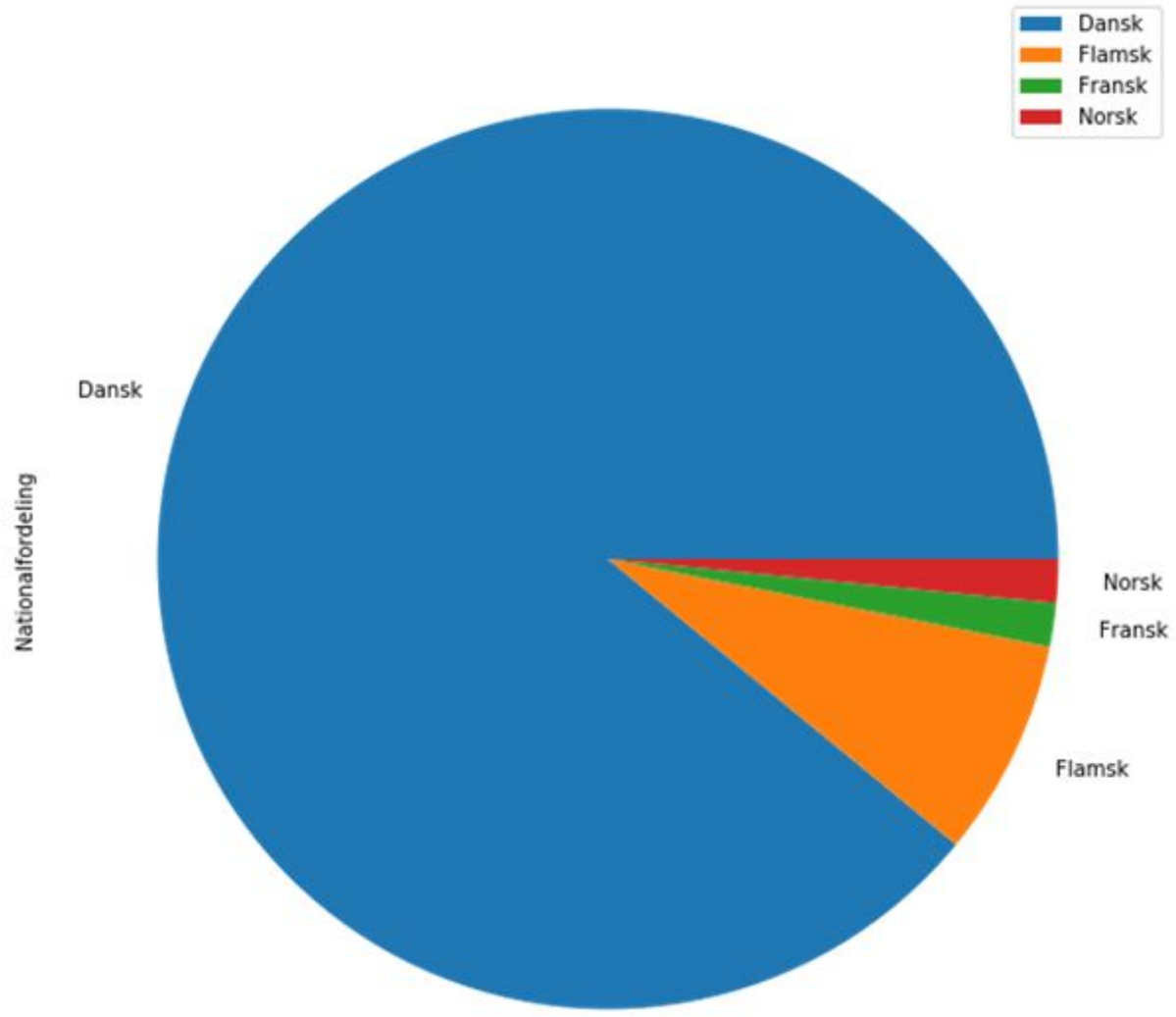
```
nattal = df['nationality'].value_counts()

natprocent = df['nationality'].value_counts(normalize=True).mul(100).round(1).astype(str) + '%'

nation = pd.DataFrame(columns=['Antal', 'Procent'], index=['Dansk', 'Flamsk', 'Fransk', 'Norsk'],
data=zip(nattal,natprocent))

print(nation)
```

	Antal	Procent
Dansk	57	89.1%
Flamsk	5	7.8%
Fransk	1	1.6%
Norsk	1	1.6%

```
natbar = df['nationality'].value_counts()[:70].plot(kind='bar')
```




```
natdf = pd.DataFrame({'Nationalfordeling': [57, 5, 1, 1],
                      'radius': [natprocent]},
                     index=['Dansk', 'Flamsk', 'Fransk', 'Norsk'])
natplot = natdf.plot.pie(y='Nationalfordeling', figsize=(10, 10))
```



Den danske dominans inden for nationalitetssektionen kan forklares ved, at Statens Museum for Kunst er et dansk museum, der agter at samle den danske kulturarv. Derfor er mange af kunstnerne af dansk ophav. Samtidig søger vi på Pe*er*, hvilket vil give et resultat, hvor populære danske navne som Peter/Peder og Petersen/Pedersen optræder. Dog udelukker vores søgning ikke, at eksempelvis en amerikansk kunstner ved navn Peter Jackson optræder. Men en amerikansk kunstner vil næppe være udstillet på Statens Museum for Kunst.

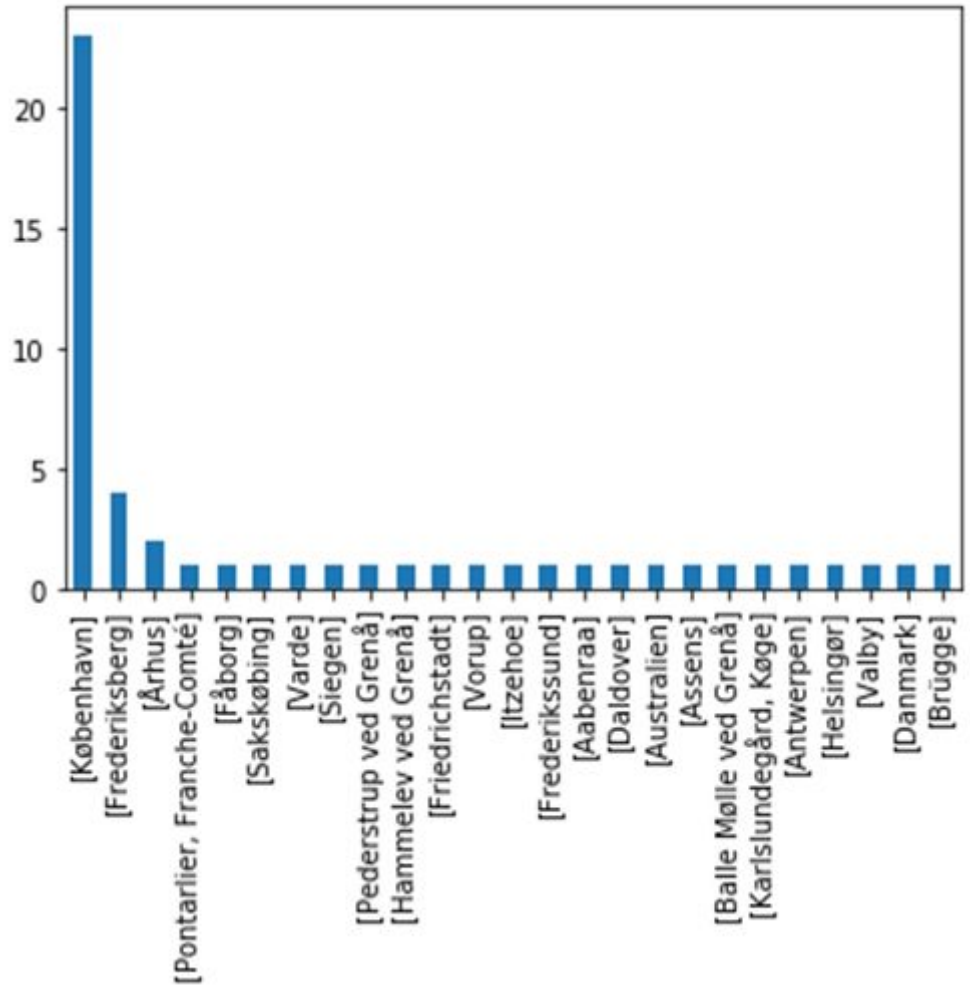
 # Fødested

```
pobtal = df['place of birth'].value_counts()

pobprocent = df['place of birth'].value_counts(normalize=True).mul(100).round(1).astype(str) + '%'
```

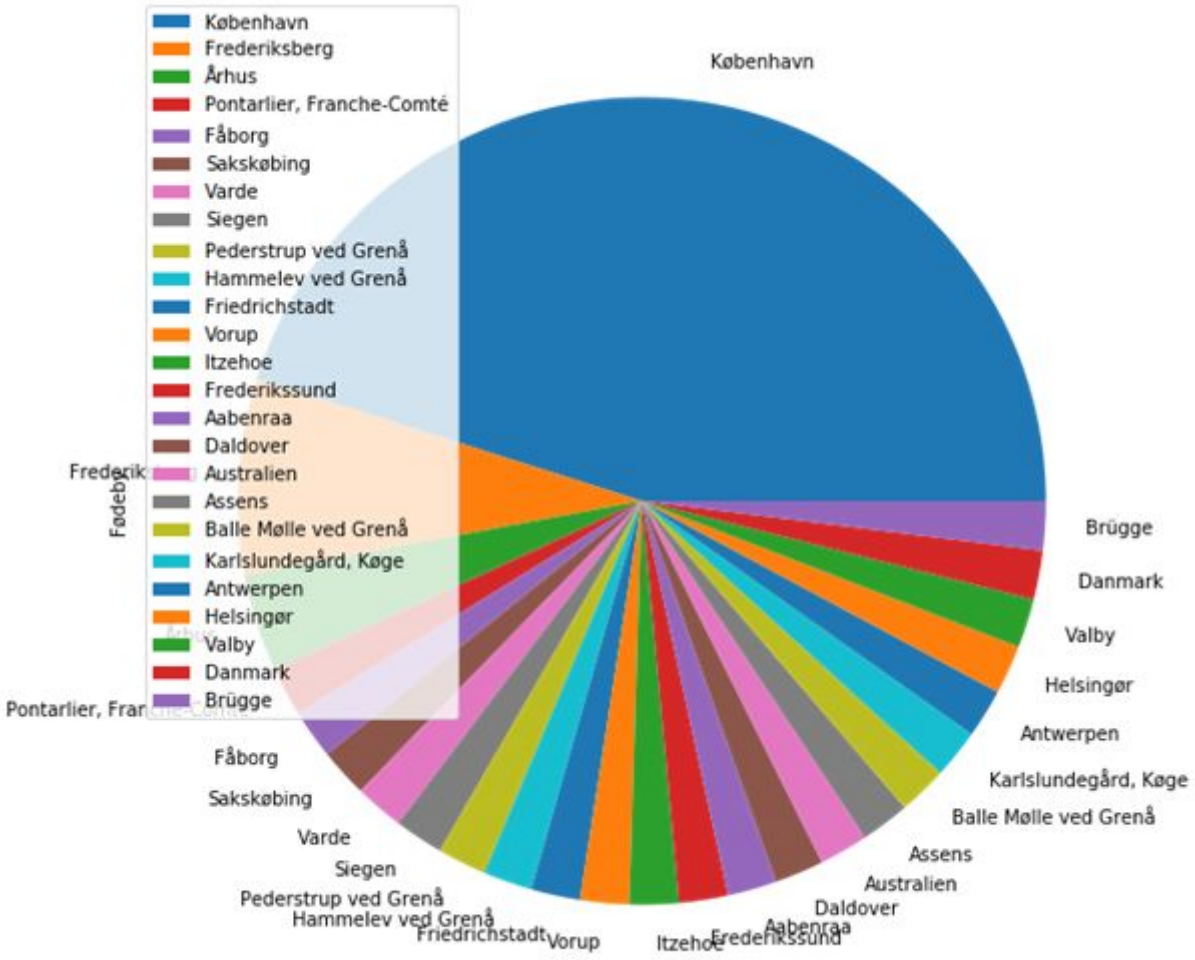
```
pob = pd.DataFrame(columns=['Antal', 'Procent'], index=['København', 'Frederiksberg', 'Århus',
'Pontarlier, Franche-Comté', 'Fåborg', 'Sakskøbing', 'Varde', 'Siegen', 'Pederstrup ved Grenå',
'Hammelev ved Grenå', 'Friedrichstadt', 'Vorup', 'Itzehoe', 'Frederikssund', 'Aabenraa', 'Daldover',
'Australien', 'Assens', 'Balle Mølle ved Grenå', 'Karlslundegård, Køge', 'Antwerpen', 'Helsingør',
'Valby', 'Danmark', 'Brügge'], data=zip(pobtal,pobprocent))

print(pob)
```

	Antal	Procent
København	23	45.1%
Frederiksberg	4	7.8%
Århus	2	3.9%
Pontarlier, Franche-Comté	1	2.0%
Fåborg	1	2.0%
Sakskøbing	1	2.0%
Varde	1	2.0%
Siegen	1	2.0%
Pederstrup ved Grenå	1	2.0%
Hammelev ved Grenå	1	2.0%
Friedrichstadt	1	2.0%
Vorup	1	2.0%
Itzehoe	1	2.0%
Frederikssund	1	2.0%
Aabenraa	1	2.0%
Daldover	1	2.0%
Australien	1	2.0%
Assens	1	2.0%
Balle Mølle ved Grenå	1	2.0%
Karlslundegård, Køge	1	2.0%
Antwerpen	1	2.0%
Helsingør	1	2.0%
Valby	1	2.0%
Danmark	1	2.0%
Brügge	1	2.0%

```
df['place of birth'].value_counts()[:70].plot(kind='bar')
```




```
pobdf = pd.DataFrame({'Fødeby': [23, 4, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],

                      'radius': [pobprocent]},

          index=['København', 'Frederiksberg', 'Århus', 'Pontarlier, Franche-Comté', 'Fåborg',
'Sakskøbing', 'Varde', 'Siegen', 'Pederstrup ved Grenå', 'Hammelev ved Grenå', 'Friedrichstadt',
'Vorup', 'Itzehoe', 'Frederikssund', 'Aabenraa', 'Daldover', 'Australien', 'Assens', 'Balle Mølle ved
Grenå', 'Karlslundegård, Køge', 'Antwerpen', 'Helsingør', 'Valby', 'Danmark', 'Brügge'])

pobplot = pobdf.plot.pie(y='Fødeby', figsize=(10, 10))
```

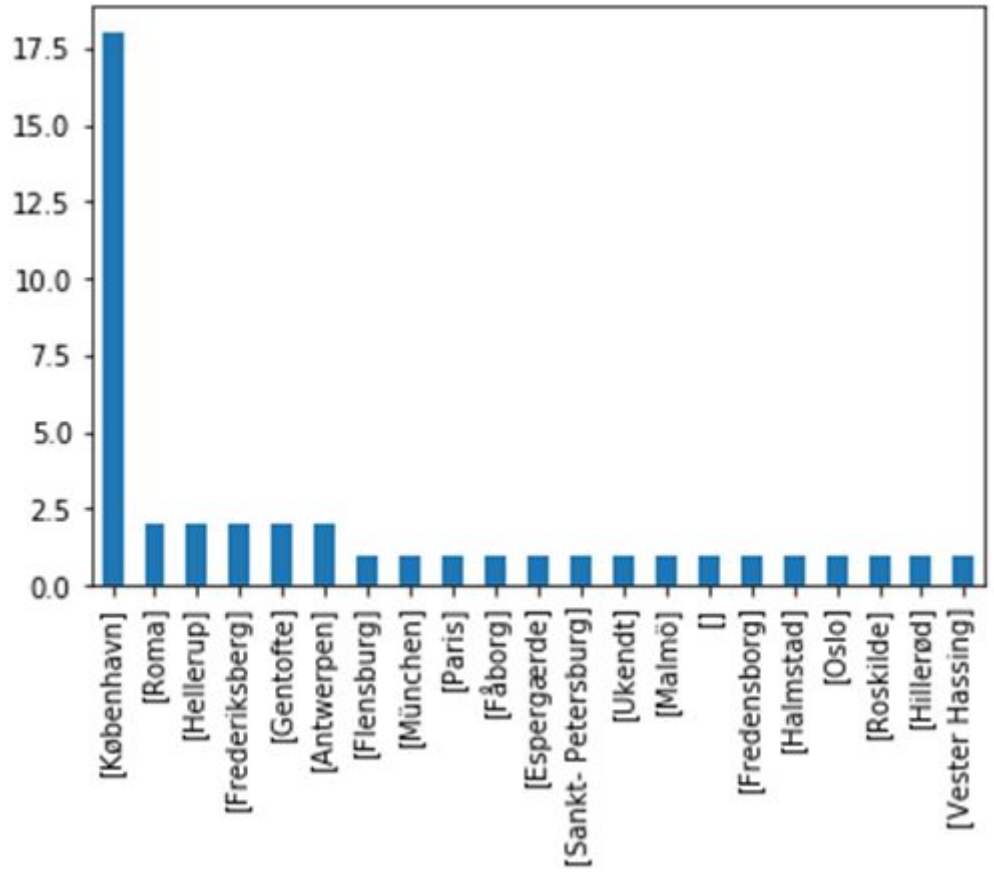


 # Dødssted

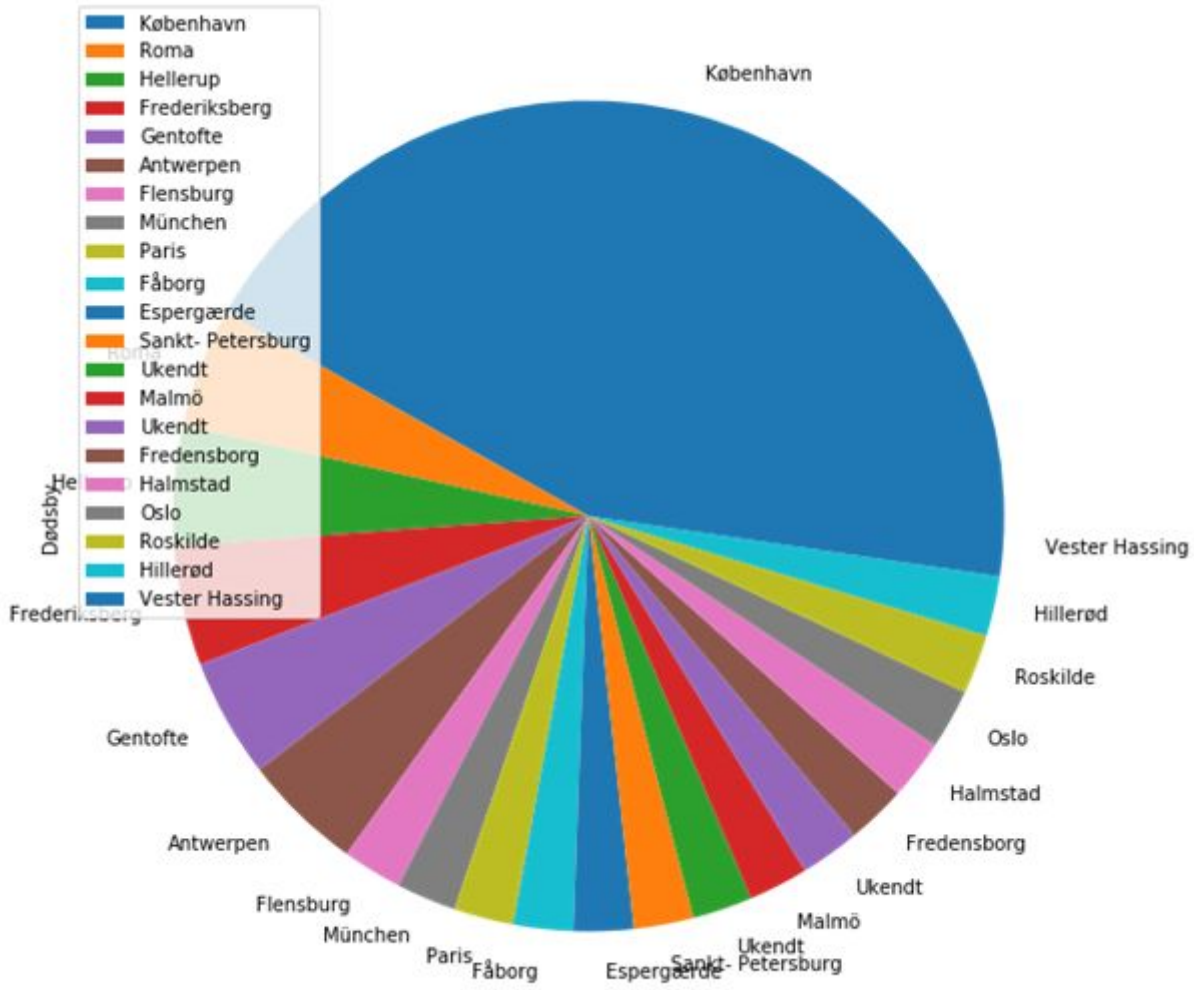
```
podtal = df['place of death'].value_counts()

podprocent = df['place of death'].value_counts(normalize=True).mul(100).round(1).astype(str) + '%'

pod = pd.DataFrame(columns=['Antal', 'Procent'], index=['København', 'Roma', 'Hellerup',
'Frederiksberg', 'Gentofte', 'Antwerpen', 'Flensburg', 'München', 'Paris', 'Fåborg', 'Espergærde',
'Sankt- Petersburg', 'Ukendt', 'Malmö', 'Ukendt', 'Fredensborg', 'Halmstad', 'Oslo', 'Roskilde',
'Hillerød', 'Vester Hassing'], data=zip(podtal,podprocent))

print(pod)
```

	Antal	Procent
København	18	41.9%
Roma	2	4.7%
Hellerup	2	4.7%
Frederiksberg	2	4.7%
Gentofte	2	4.7%
Antwerpen	2	4.7%
Flensburg	1	2.3%
München	1	2.3%
Paris	1	2.3%
Fåborg	1	2.3%
Espergærde 1	2.3%	
Sankt- Petersburg	1	2.3%
Ukendt	1	2.3%
Malmö	1	2.3%
Ukendt	1	2.3%
Fredensborg	1	2.3%
Halmstad	1	2.3%
Oslo	1	2.3%
Roskilde	1	2.3%
Hillerød	1	2.3%
Vester Hassing	1	2.3%

```
podbar = df['place of death'].value_counts()[:70].plot(kind='bar')
```




```
poddf = pd.DataFrame({'Dødsby': [18, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],

                      'radius': [podprocent]},
          index=['København', 'Roma', 'Hellerup', 'Frederiksberg', 'Gentofte', 'Antwerpen', 'Flensburg',
'München', 'Paris', 'Fåborg', 'Espergærde', 'Sankt- Petersburg', 'Ukendt', 'Malmö', 'Ukendt',
'Fredensborg', 'Halmstad', 'Oslo', 'Roskilde', 'Hillerød', 'Vester Hassing'])

podplot = poddf.plot.pie(y='Dødsby', figsize=(10, 10))
```



Refleksion over SMK's API

Ved arbejdet med SMK's API kunne der dokumenteres flere problemer bestående af at selve APIet ikke virkede intuitiv, hvis man ikke har tidligere er stiftet bekendtskab med denne type API. Vi fandt det besværligt at arbejde med, da vi skulle undersøge både hvert enkelt kolonnes type af indhold og vurdere indholdets kvalitet gennem stikprøver f.eks. ved kunstnernes fødselsdage, hvor der var angivet 3 datoer for hvert enkelt kunstner; *birth_date_start, birth_date_end* og *birth_date_prec*, der ofte ikke var den samme dato. Hvis den egentlige dato var ukendt, ville det ved *birth_date_star*t stå den første dato for det år, man antog ville være den tidligst mulige, samtidig med at der ved *birth_date_end* ville stå den sidste dato for det år, man antog ville være den senest mulige dato. Derudover fremgik der også *birth_date_prec*, hvor der i nogle tilfælde stod flere årstal, men i andre tilfælde bare stod som ukendt. Ud fra disse data, ville det fremgå som om nogle kunstnere havde fødselsdag hver eneste dag i flere år. Jo flere felter, for den samme type af data, der er, jo mere besværligt er det at indtast nye data. De mange forskellige felter for fødselsdage, giver større

chancer for ukomplette data, da de kan forvirre både for dem der skal indtaste data, men også for dem, der skal arbejde videre med disse data.
Det er vigtigt at et API har relevante data med. Da vi undersøgte vores datasæt fra SMK i forhold til Dublin Core's standarder, kunne vi se at der var flere af Dublin Core's kolonner eks. Publisher, der ville være irrelevant. At tilføje de ekstra kolonner, ville ikke gavne i forhold til at lave brugbare analyser af indholdet, da det kunne skabe flere forstyrrelser.

Hvorvidt metadataet om de forskellige materialer bliver opdateret, kan være svært at være sikker på da, det skal gøres manuelt. Hvis et maleri rykkes, sælges, udlånes eller lignende, skal det manuelt ind og ændres, derfor er der en sandsynlighed for at dette ikke er blevet gjort, eller vil blive gjort med eftervirkning.
En stor del af de angivne data i vores datasæt var mangelfulde, altså skrevet som ukendte eller upræcise datoer eller steder. Dette kan også gøre det svært eller umuligt at arbejde med specifikke data, der der er for mange mangler.
I APIet over artworks, er der meget detaljeret data, alt fra målene på et maleri, dimensionerne på en skulptur, hvis kunstværket har været flere år undervejs, er både start og slut år registreret. Til sammenligning indeholder APIet ikke samme uddybende datamængde omkring kunstneren. Her kunne man netop ved hjælp af det elektroniske, linke kunstneren til hans/hendes værker, og opdele det efter perioder, hvor kunstneren havde forskellige stilarter eller teknikker, Claude Monét's værker med åkander kunne under Monét være samlet i en underkategori, således at APIet kunne underbygge den kunstneriske process og stilarter.

Den åbne tilgang til SMK's samling, kan give muligheder for andre kunstmuseer til at få adgang til værker der er i SMKs magasiner. Potentielt kunne et mindre museum søge i API'et og finde værker til en udstilling om broer, ved blot at søge på broer, og derved få mange forskellige kunsters fortolkninger af ordet og genstanden bro. De forskellige museers opslagskataloger omkring deres værk samlinger har førhen været fysiske og ikke alment tilgængelige. De ældre fysiske opslagskataloger har omhandlet f.eks. en specifik kunstner eller et museums samling. Disse opslagskataloger har ikke nødvendigvis kunne opdateres omkring udlånte materialer. Dette kan gøre processen i at finde værker til en bestemt udstilling som f.eks. Nan Dalsgaards udstilling om "Sommerlandet Danmark" besværlig, da der skal ledes mange forskellige steder efter relevant materiale. Med flere åbne samlinger og API'er vil det være muligt at søgninger efter særlige emner, vil blive markant lettere.

Litteraturliste Portfolio 3

Schultz, S. (2011). *C.C. Peters* i: **Den store danske.dk** (2019) Gyldendal, set online 9. december 2019 (http://denstoredanske.dk/Dansk_Biografisk_Leksikon/Kunst_og_kultur/Billedkunst/Billedhugger/C.C._Peters)

HACK4DK, (U/Å). *HACK4DK - Hack din kulturarv,* set online 9. december 2019 *https://www.smk.dk/event/hack4dk-2019/*

Sanderhoff, M., & Edson, M. (2014). *Sharing is caring: openness and sharing in the cultural heritage sector.* Kbh: Statens Museum for Kunst.

SMK, (U/Å). *SMK open/Making art available for everyone.* Kbh: Statens Museum for Kunst. *https://www.smk.dk/wp-content/uploads/2018/06/Projektbeskrivelse_SMK-Open.pdf*