# Københavns Universitet
# Grundlæggende Data Science - Individual Pass/Fail
# Exercise 1

Victor Vangkilde Jørgensen - kft410

February 10, 2025

# Contents

**1 Part 1: Regular expression warmup** **3**
1.a Danish CPR numbers . . . 3
1.b . . . 3

**2 Part 2: Processing the FakeNewsCorpus data set** **3**
2.a . . . 3
2.b . . . 3
2.c . . . 3
2.d . . . 3
2.e . . . 4

**3 Part 3: Descriptive frequency analysis of the data** **4**
3.a . . . 4
3.b . . . 4

# 1 Part 1: Regular expression warmup

**In these questions, you are expected to use functionality from the Python regular expression module: re.**

**1.a** **Danish CPR numbers consist of a six-digit date (DDMMYY), followed by a 4 digit identifier (IIII). Using the re module, the task is to write and apply a regular expression that matches any CPR number, either in the DDMMYYIIII format, or in the DDMMYY-IIII format. The regular expression should contain four groups, such that the DD, MM, YY, and IIII parts can be extracted after matching.**

**1.b** **The 4-digit identifier together with the last two digits of the date (i.e. the year), encodes in which century a person is born, using the following system**

| IIII | YY | Born in century |
|---|---|---|
| 0001-3999 | 00-99 | 1900 |
| 4000-4999 | 00-36 | 2000 |
| 4000-4999 | 37-99 | 1900 |
| 5000-8999 | 00-57 | 2000 |
| 5000-8999 | 58-99 | 1800 |
| 9000-9999 | 00-36 | 2000 |
| 9000-9999 | 37-99 | 1900 |

**Note that in this table, both numbers in a range are included (e.g. 00-99 includes both 00 and 99).**

**Write a function that returns the relevant century based on the information in the table above, and returns either 1800, 1900 or 2000 (as an integer).**

# 2 Part 2: Processing the FakeNewsCorpus data set

**This part focuses on getting the FakeNewsCorpus dataset in a reasonable shape, and doing some initial exploration of the data. You will be working on a sample of the FakeNewsCorpus.**

**2.a** **Download the CSV onto your computer**

**2.b** **Read the CSV file into memory**

**2.c** **Manually inspect the data to get an idea of potential problems of the data structure and representation that need to be fixed.**

**2.d** **Clean the data. First, you'll try to do this manually, by writing our own cleanText() function that uses regular expressions. The function should take raw text as input and return a version of the text with the following modifications:**

- **all words must be lowercased**
- **it should not contain multiple white spaces, tabs, or new lines**
- **numbers, dates, emails, and URLs should be replaced by "<NUM>", "<DATE>", "<EMAIL>" AND "<URL>", respectively. Note that replacing dates with <DATE> is particularly tricky as dates can be expressed in many forms. You may just choose one or a few common date formats present in the data set and only replace those. (Be careful about tokenizing <> symbols because these are punctuation in most Tokenizers).**

**2.e** **Now, try to use a library for cleaning the data. The clean-text module provides out-of-the-box functionality for much of the cleaning you did in the previous step (pip install clean-text). Use it to implement the same cleaning steps as in your own cleanText() implementation.**

## 3 Part 3: Descriptive frequency analysis of the data

**Now, you will do some simple exploration of the data to understand the effect of preprocessing conducted in Part 2, namely to the vocabulary, and the frequency distribution of the vocabulary across documents.**

**3.a** **Calculate the number of unique words in the data after preprocessing and compare it to the number of unique words before preprocessing (i.e. in the raw data).**

**3.b** **Calculate how frequently each of these words is used in the dataset. Sort this list, so that the most frequent word appears first, and then use Python's matplotlib library to plot a barplot where the 50 most frequent words appear on the x-axis, and their frequency appears on the y-axis.**