Datasets and handling data in Python

Objective: The objective is to you acquainted with the exercise format of the course. Additionally, the aim is to familiarize yourself with the standard dataformat used in the course. Upon completing this exercise it is expected that you:

- Understand the format of the exercises and how the exercises are related to the reports.
- Can import data into Python and represent the data in course $\mathbf{X}$, $\mathbf{y}$ format.
- Can do common preprocessing steps for datasets.
- Understand the *bag of words* representation for text documents including filtering methods based on removal of stop words and stemming.
- Started the process of selecting a proper dataset for use in the your project work (for the reports).

Material:

PYTHON Help: You can get help in your Python interpreter by typing `help(obj)` or you can explore source code by typing `source(obj)`, where `obj` is replaced with the name of function, class or object.
Furthermore, you get context help in VS Code after typing function name or namespace of interest. In practice, the fastest and easiest way to get help in Python is often to simply Google your problem. For instance: `"How to add legends to a plot in Python"` or the content of an error message. In the later case, it is often helpful to find the *simplest* script or input to script which will raise the error.

Discussion forum: You can get help on our online discussion forum:
https://piazza.com/dtu.dk/spring2025/02450

Software installation: Extract the Python toolbox from DTU Inside and install the `dtuimldmtools` package if you have not already done so (see Exercise 0). Open your Python IDE (e.g. VSCode) and make sure to activate/select the environment in which you installed the `dtuimldmtools` package. Remember the purpose of the exercises is not to re-write (all) the code from scratch but to work with the scripts provided in the directory `<base-dir>/02450Toolbox_Python/Scripts/`
Representation of data in Python:

	Python var.	Type	Size	Description
	`X`	numpy.array	$N \times M$	Data matrix: The rows correspond to N data objects, each of which contains M attributes.
	`attributeNames`	list	$M \times 1$	Attribute names: Name (string) for each of the M attributes.
	`N`	integer	Scalar	Number of data objects.
	`M`	integer	Scalar	Number of attributes.
Classification	`y`	numpy.array	$N \times 1$	Class index: For each data object, `y` contains a class index, $\mathtt{y}_n \in \{0, 1, \ldots, C-1\}$, where C is the total number of classes.
	`classNames`	list	$C \times 1$	Class names: Name (string) for each of the C classes.
	`C`	integer	Scalar	Number of classes.

1.1 How to do the exercises

During exercises in the course, you will go through an exercise document like this one. The exercises are primarily centered around running, understanding and modifying a series of scripts provided in the 02450 Toolbox along with relevant pen and paper problems. The exercise descriptions guide you through the scripts often focusing on modifying minor parts of the code to understand the most important part of the data and the model/algorithm and their properties. We do not ask you to implement whole algorithms from scratch, as this would be difficult to achive within the scope of the course, but some exercise will include specific optional challenges related to implementations details for the ambitious/interested students. The scripts are also the basis for your work in the reports, where you will be able to re-use large parts of the code. For the reports, you will tailor the code to your dataset and problem.

The exercises are structured as smaller numbered sections. When a certain section concerns a particular script, it will be stated and their number will match. For instance, the first script you will run (in a little while) is called `ex1_5_1.py` and corresponds to the section 1.5.1 in this document.

1.2 Getting started with Python

We assume that you have a working Python IDE set up. If that is not the case, complete the pre-exercise (Exercise 0) before proceeding. If you have already done the optional Exercise 0, you can skip the next section ("Installing the 02450 Toolbox").

We will assume you have the **Anaconda** Python distribution or another Python distrbution with Python version 3.8-3.11 (see Exercise 0 for details). Additionally, in the following, it will be assumed VSCode is used to run python commands and edit Python files; however, you are free to use another IDE if you prefer.

1.3 Installing the 02450 Toolbox

The course will make use of several specialized scripts and toolboxes not included with Python. These are distributed as a toolbox and a Python package which need to be installed.

1.3.1 The scripts/code you will primarily focus on during the exercises are located on DTU Learn in a zip-file (`02450Toolbox_Python.zip`) that you need to download.

Identify the location of the downloaded zip-file, unzip it, and locate the folder called `02450Toolbox_Python`. Copy the 02450Toolbox_Python folder (and content) to the working directory that you use for the course.

```
<base-dir>/02450Toolbox_Python/Scripts/   # Scripts for exercises
```

For the exercises, you should work on the example scripts in `<base-dir>/02450Toolbox_Python/Scripts/` (notice the scripts are labelled according to exercise number) and not try to write the scripts from the bottom up.

1.3.2 Addtionally, you need to install a custom Python package with dedicated 02450 functions and utilities called `dtuimldmtools` These are distributed as a course-specific Python package `dtuimldmtools` which needs to be installed. Once the Python environment has been created, one can install the package using the following command in the terminal. Make sure to install the package inside the specific Python (virtual) environment you have created, e.g., using conda as shown here:

```
conda activate <my-env>
pip install dtuimldmtools
```

We recommend you restart restart VSCode at this point.

1.3.3 In VS Code, open the folder `<base-dir>/02450Toolbox_Python/Scripts/` and check that you can execute the `check_installtion.py` file without errors.

1.4 Representation of data in Python

We will use a standard data representation throughout the course. Using this representation makes it easy to apply the various tools in the 02450 Toolbox on a new dataset. Once you have a given dataset in the standard format, the scripts will all be set up to work with it correctly.

An overview of the format is presented for Python in this table:

	Python var.	Type	Size	Description
	`X`	numpy.array	$N \times M$	Data matrix: The rows correspond to N data objects, each of which contains M attributes.
	`attributeNames`	list	$M \times 1$	Attribute names: Name (string) for each of the M attributes.
	`N`	integer	Scalar	Number of data objects.
	`M`	integer	Scalar	Number of attributes.
Classification	`y`	numpy.array	$N \times 1$	Class index: For each data object, `y` contains a class index, $\mathtt{y}_n \in \{0, 1, \ldots, C-1\}$, where C is the total number of classes.
	`classNames`	list	$C \times 1$	Class names: Name (string) for each of the C classes.
	`C`	integer	Scalar	Number of classes.

1.5 Loading data

Before we can begin to do machine learning, we need to load the data. Datasets are distributed as various types of files, and a few common ones will be shown here for future reference to be used once you have to load your own dataset.

Once we have loaded a dataset, we often need to process the data before the format fits our needs. For this course, in particular, this mostly means putting the dataset in the $\mathbf{X}$, $\mathbf{y}$-format shown above. The machine learning algorithms you will use needs the data to be in a numerical format, so we will also go through how to convert data which is in a text format into a numerical format.

Lastly, we will also go through a few tasks that often need to be handled before we can load some dataset.

For today's exercises you need to add a package to your Python environment. The additional package is for importing data from excel spreadsheets. Please make sure that you have installed the following packages (you can follow the guidelines at the corresponding websites):

- Excel file data extraction (xlrd package):
  `https://xlrd.readthedocs.io`

The websites provide documentation of the packages. Note if you use the Anaconda Python distribution these packages may already be added, use `conda list` in the terminal for a list of installed packages.

We consider the Iris flower dataset, or Fisher's Iris dataset, is a multivariate dataset introduced by Sir Ronald Aylmer Fisher (1936) for the problem of classifying Iris flower types. It is sometimes called Anderson's Iris dataset because Edgar Anderson collected the data to quantify the geographic variation of Iris flowers in the Gaspé Peninsula. The dataset consists of 50 samples from each of three species of Iris flowers (Iris setosa, Iris virginica and Iris versicolor). Four variables were measured from each sample: the length and the width of sepal and petal, in centimetres. Based on the combination of the four variables, Fisher developed a model to distinguish the species from each other. It is used as a typical test for many other classification techniques (see also `http://en.wikipedia.org/wiki/Iris_flower_data_set`). The data has been downloaded from `http://archive.ics.uci.edu/ml/datasets/Iris`.

A simple format of storing data is the comma-separated values-file format (or CSV). In such files, a sample or an observation is a line in a text document, and the document then has as many lines (or rows) as there are samples. The attribute values for an observation is written within one line, separated by (usually) a comma or a tab-character in a consistent order. This order is usually defined in a header (the first line of the file), which has a designation of the variable name in some format.

1.5.1 Inspect and run the script `ex1_5_1.py` to see how to load the Iris data from a CSV-file and put it into the standard format. Since the class label (the flower species) are stored as text (or strings), we convert them into a numerical value.

  The data files are installed with the `dtuimldmtools` package. The location of the data file, `iris.csv`, is stored in the `filename` variable and displayed in the terminal after running `ex1_5_1.py`. Inspect the file `iris.csv` using a simple text editor (e.g. for Windows "Notepad" or for MacOS "TextEdit").

1.5.2 Sometimes datasets are distributed as Excel-files (`.xls(x)`). Inspect the script `ex1_5_2.py` to see how to load the same Iris data, when it has been stored as an Excel-file (open `iris.xls` in the data folder to have a look at the file).

1.5.3 Other times, and especially in this course, data is stored as MATLAB files (`.mat`). Inspect `ex1_5_3.py` to load the Iris data from `iris.mat` in the data folder.

1.5.4 In the examples up until now, we have handled the data in the Iris dataset as if to solve a classification problem. We could say that the *primary* machine learning modelling aim is to classify the species of Iris flower based on the petal and sepal dimensions. However, we could also use the dataset to illustrate how to do regression without needing to use a whole different dataset. We would achieve this by e.g. trying to predict either of the petal (or sepal) dimensions based on the remaining dimensions, for instance. This changes how we define our $\mathbf{X}$,$\mathbf{y}$-format. Inspect `ex1_5_4.py` to see how to cast the Iris dataset into a regression problem. In the script, we will set up the $\mathbf{X}$,$\mathbf{y}$-format such that we are predicting the petal lengths form the other available information. Notice that we change how we use the information of the class label from before (the species information). Instead of storing it as a single variable, we have now used a "one-out-of-K encoding", since it is a categorical variable—we will return to various types of variables when we go through chapter 2 in the book (where one-out-of-K-encodings are described in section 2.4.1).

1.5.5 While the Iris dataset is a real dataset, it is a very clean and easy to work with dataset. Usually, data is a bit messier, and we will consider a toy dataset that has some common issues. Often, the description of "real-world" data is stored along with the data in some form of a text file. Have a look at the folder `messy_data` in the data folder,

  and read more about the toy dataset—notice that there is a `README.txt` file.

Inspect the data in `messy_data.data` and try to identify some issues (use e.g. simple text editor as before) Afterwards, inspect `ex1_5_5.py` to see how the dataset can be stored in the desired representation.

1.6 The document-term matrix (consider skipping this part for now and returning to it at a later time)

An important area of research in machine learning and data mining is the analysis of text documents. Here, important tasks are to be able to search documents as well as group related documents together (clustering). In order to accomplish these tasks the text documents is converted into a format suitable for data modeling. We will use the *bag of words* representation. Here, text documents are stored in a matrix $\mathbf{X}$ where x_{ij} indicate how many times word j occurred in document i.

Suppose that we have 5 text documents, each containing just a single sentence.

Document 1:	The Google matrix P is a model of the internet.
Document 2:	P_{ij} is nonzero if there is a link from webpage i to j.
Document 3:	The Google matrix is used to rank all Web pages.
Document 4:	The ranking is done by solving a matrix eigenvalue problem.
Document 5:	England dropped out of the top 10 in the FIFA ranking.

1.6.1 Propose a suitable *bag of words* representation for these documents (use pen and paper). You should choose approximately 10 key words in total defining the columns in the document-term matrix and the words are to be chosen such that each document at least contains 2 of your key words, i.e. the document-term matrix should have approximately 10 columns and each row of the matrix must at least contain 2 non-zero entries.

1.6.2 In practice, the above procedure is carried out automatically, see the script `ex1_6_2.py`. We will use an `sklearn` function from the feature extraction-module, called `CountVectorizer` to generate a document-term matrix and to convert it into the format described in the beginning of the exercise (Representation of data in Python).

Script details:

- *Make sure that you have the* `sklearn`*-package installed.*
- *Read more about the CountVectorizer using:*
`help(CountVectorizer)`
after it has been imported from `sklearn`*.*

Compare the generated document-term matrix to the one you generated yourself.

Stop words are words that one can find in virtually any document. Therefore, the occurrence of such a word in a document does not distinguish the document from other documents. The following is the beginning of one particular stop word list:

a, a's, able, about, above, according, accordingly, across, actually, after, afterwards, again, against, ain't, all, allow, allov, ahnost, alone, along, already, also, although, always, am, among, amongst, an, and, another, any, anybody, anyhow, anyone, anything, anyway, anyways, anywhere, apart, appear, appreciate, appropriate, are, around, as, aside,ask, ...

When forming the document-term it is common to remove these specified stop words.

1.6.3 The generated document-term matrix contains words that carry little information such as the word “the”. We will remove these words as they can be interpreted as “noise” carrying no information about the content of the documents. Compute a new document-term matrix with stop words removed using `ex1_6_3.py`

Script details:

- *A list of stop words is stored in the file* `../Data/stopWords.txt`.
- *Once the stop words are loaded, they can be parsed to the* `CountVectorizer` *by parsing it using the keyword* `stop_words`.

Inspect the document-term matrix: How does it compare to your original matrix?

Stemming denotes the process for reducing inflected (or sometimes derived) words to their stem, base or root form. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. Clearly, from the point of view of information retrieval, no information is lost in the following stemming reduction:

$$\left.\begin{array}{l} \textit{computable} \\ \textit{computing} \\ \textit{computed} \\ \textit{computational} \\ \textit{computation} \end{array}\right\} \rightarrow \textit{comput}$$

1.6.4 Document 3, 4 and 5 have the word “rank” in common. However in document 4 and 5 this word is stored as a the separate word entry “ranking” in the document-term matrix whereas in document 3 it is stored as the word entry “rank” . As such, the document-term matrix does not indicate that document 3, 4 and 5 share the word “rank”. By the use of stemming we can obtain a matrix that indicate that the word “rank” appears in all 3 documents. Enable stemming and compute a new document-term matrix using the script `ex1_6_4.py`.

Script details:

- *Make sure you have the* `nltk`*-package installed.*
- *You can stem verbs using the a* `PorterStemmer`*. Once the PorterStemmer is made, try writing:* `stemmer.stem('computational') == stemmer.stem('computable')`.

Inspect the document-term matrix: How does it compare to your original matrix?

Based on our document-term representation we can now make simple searches (queries) in our documents based on some form of similarity measure between our query vector and document-term representation. Lets say we want to find all documents that are relevant to the query “**solving** for the **rank** of a **matrix**.” This is represented by a query vector, $\boldsymbol{q}$, constructed in a way analogous to the document-term matrix, $\boldsymbol{X}$:

$$\begin{array}{cc} & \begin{array}{ccccccccccccccccc} \text{drop} & \text{eigenvalu} & \text{england} & \text{fifa} & \text{googl} & \text{internet} & \text{link} & \text{matrix} & \text{model} & \text{nonzero} & \text{page} & \text{problem} & \text{rank} & \text{solv} & \text{top} & \text{web} & \text{webpag} \end{array} \\ \boldsymbol{X} = & \left[\begin{array}{ccccccccccccccccc} 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{array}\right] \end{array}$$

$$\boldsymbol{q} = \left[\begin{array}{ccccccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{array}\right]$$

1.6.5 We will use the *cosine distance* as a measure of similarity between the i'th document $\boldsymbol{x}_i$ and the query vector $\boldsymbol{q}$, i.e. $\cos(\boldsymbol{q}, \boldsymbol{x}_i) = \frac{\boldsymbol{q}\boldsymbol{x}_i^\top}{\|\boldsymbol{q}\|\|\boldsymbol{x}_i\|}$. (We will later in the course learn much more about measures of similarity). Compute the cosine similarity between each document and the query using a) pen and paperv (i.e. compute the inner products between the relevant vectors), and b) `ex1_6_5.py`, and show that Document 4 is most similar to the query.

Script details:

- *You can extract a document (row of the* `X` *matrix) using the command* `x=X[i,:]` *where* `i` *is the index of the document.*
- *Numpy matrices and arrays can be transposed using notation* `m.T` *or* `m.transpose()`*.*
- *Dot products between two row vectors can be computed as* `numpy.dot(q,X.T)` *(or simply* `q@X.T`*).*
- *The norm of a vector can be computed using the function* `numpy.linalg.norm()`*.*

Explain what documents, according to our similarity measure, are most related to the query.

1.6.6 (OPTIONAL) If you find text processing exciting, read more about Natural Language Processing toolkit. Here is a good place to start:
`http://www.nltk.org/book/`

As part of project 1, you will have to think about how various machine-learning tasks can be carried out for the dataset chosen and this exercise will briefly touch upon this. Discuss the following questions:

1.7 Selecting a dataset and tasks for the report

You are now able to select a dataset and address the following tasks for the report.

The course will include two written group reports. The two reports will cover the first two sections of the course:

- Data: Feature extraction, and visualization
- Supervised learning: Classification and regression

Each group will find one dataset they will use for the reports. This can either be your own dataset or a dataset you find yourself. Additional details about where you can find a dataset and formal requirements can be found in the **Finding a dataset for the reports** `.pdf` file on DTU Learn.

As part of project 1, you will have to think about how various machine-learning tasks can be carried out for the dataset chosen and this exercise will briefly touch upon this. Discuss the following questions:

1. **A description of your data set.**
    - Explain the overall problem of interest and the associated data.
    - Provide a reference to where you obtained the data.
    - Summarize previous analysis of the data. (i.e. go through one or two of the original source papers and read what they did to the data and summarize their results).

- You will be asked to apply (1) classification and (2) regression on your data in the next report. For now, we want you to consider how this should be done. Therefore:

  - Explain, in the context of your problem of interest, what you hope to accomplish/learn from the data using these techniques?
  - Explain which attribute you wish to predict in the regression based on which other attributes?
  - Which class label will you predict based on which other attributes in the classification task?
  - Explain if you need to transform individual attribues in order to carry out these tasks (e.g. centering, standardization, discretization, log transform, etc.) and how you plan to do this.

  One of these tasks is likely more relevant than the rest and will be denoted the **main machine learning aim** in the following.

2. **A detailed explanation of the attributes of the data.**

   - Describe if the attributes are discrete/continuous and whether they are nominal/ordinal/interval/ratio.
   - Give an account of whether there are data issues (i.e. missing values or corrupted data) and describe them if so and how you will handle them.

References