Essentials for Data Science: Exam 2024/25 EXAMPLE (Part 2 of 2: programming)

- Date: Friday, 13th June 2025
- Duration: approx. 09:45-12:00

Questions and input data

- The questions are provided in this document.
- In Brightspace there are 4 files which need to be downloaded:
    - Python notebook: `exam.ipynb`.
    - Data files needed for the questions: `names.txt`, `grades.json` and `apidae.csv`.

Solving the exam

- Answer the questions in the `exam.ipynb` notebook. Fill the provided cells with your answers. Please do not add or remove cells.
- You are allowed to use the internet to search for information.
- You are **not allowed** to use AI techniques to solve the questions.
- You are allowed to use books or personal notes.
- During the exam, any communication with other individuals, by any means, is not allowed. No chat, email, or similar applications can be visible on your screen at any moment.

Submission

- ~~Before the end of the exam please submit the edited `exam.ipynb` file to Brightspace. Make sure to save the file before submitting it. Submission process may last a few minutes, so please start early.~~
- **Your name or your student number must not appear in the notebook, nor in its filename.** Brightspace will keep track of your files and provide them in a random order for anonymous grading.

Question 1: [12p] Primitive string encoding and decoding.

Introduction

Study the following `encode(txt, div)` function.
The function returns a list of numbers, representing an encoded version of the text provided in the `txt` argument.
The integer `div` plays a role of a secret key.
The function goes through each character of `txt`: its code (`ord`) and its position get encoded into a single number.

```
# ---------------------------------------------------
def encode(txt, div):
    return sorted([(idx + ord(l)*div) for idx, l in enumerate(txt)])

encodedNums = encode(txt="Encrypt me, then decrypt me!", div=349)
print(encodedNums)
# ---------------------------------------------------
```

For illustration, let's study coding of a single character:

- There is a letter c at the index 2 of the `txt` argument.
- The letter c has the code `ord('c') == 99`.
- For `div=349` the letter c at index 2 leads to the number `2 + 349*99 == 34553`. Run the code and find that number in the output.

Assume that the character codes are integers in the range of 0 to 127 only.

Task A: implement the decode function

Now, implement the function `decode(encodedNums, div)`.
As the arguments the function should take a list of encoded numbers in `encodedNums` and a secret key in `div`.
The function should return the decoded, original text as a `str` object.
Potential hints: `//`, `%`, `chr()` and `join()`.

Task B: test the decode function

- Try two short English sentences (20-30 characters long) of your choice and encode them with the encode function, using `div >= 300`.

- Decode the encoded numbers with the `decode` function.

- Write a test that checks whether the original text is the same as the decoded text. Raise an exception if the test fails.

Task C: find the secret key and message

For the encoded list given below, find the original English sentence (the secret code is one of 503, 509, 659, 797, 821):

```
# ----------------------------------------------
print( decode(
    encodedNums = [25508, 25511, 25519, 25526, 25529, 46240,
        50238, 63776, 65380, 69339, 77311, 78114, 80506, 80509,
        82889, 82907, 83690, 87691, 88487, 88490, 90871,
        90882, 91661, 92455, 92462, 92463, 92470, 96454],
    div=???  # Find the secret key.
) )
# ----------------------------------------------
```

Question 2: [12p] Calculating average grades.

Task overview

Several people gave presentations and several experts graded each presentation. The task is to print a table with average grades for each presenter.

Input files

- `names.txt`: contains the names of the people registered for the presentations (one full name per line, no duplicates).
  You may need to use `strip()` to remove the newline character at the end of each name.
- `grades.json`: contains all grades given to all presentations.
  The file represents a list, and each list element represents one given presentation.
  Each list element is a dictionary and provides the `name` of the presenter and the `grades` given by the experts:
    - `grades`: are lists of numbers between 1 and 10.
    - `name`: a string, for the registered people it is identical to a name from the `names.txt` file.

Notes:

- Some registered people did not present or did not receive any grades: for them the average grade should be nan.
- A few presentations were given twice, so there can be elements with the same presenter name: the average grade should be calculated from all available grades for each name.
- There were some unregistered people providing presentations.

Output

Print the following table:

- The rows:
  - One row represents a single person.
  - Row order:
    - First, registered people in exactly the same order as in `names.txt`.
    - Then, the unregistered people (in any order).
- The columns, separated by tabs, should be as follows (order is important):
  - `N`: the number of grades received by the person (a non-negative integer),
  - `Mean`: the average grade (a number, rounded to 1 decimal place),
  - `Extra`: whether the person was registered or not (`True` or `False`),
  - `Name`: the name of the person (a string).
- The header row should be included.

Here is an example output (3 top and 3 bottom rows are shown):

```
N     Mean  Extra Name
0     nan   False Coenraad Dąbrowski
0     nan   False Meindert Visser
3     7.0   False Susanna Lewandowski
...
2     6.8   False Konstanty Lewandowski
0     nan   True  Susanna Hoffmann
2     8.5   True  Roseann Sousa
```

Question 3: [12p] Elapsed days.

Task

1583 was the first full year in which the Gregorian calendar, which introduced new rules for leap years, was used.
Implement the function `elapsedDays(startYear, startMonth, startDay, endYear, endMonth, endDay)` which returns the number of days that elapsed between the start date and the end date.
The dates are given as integers: the year, the month, and the day.
The function should raise a `ValueError`: if the end date is before the start date, or if the dates are not valid (e.g., February 30th).
The function should have a short docstring. Other comments are not graded.

You are NOT ALLOWED to use any date-time libraries.

Hint

A *possible* algorithm will be to implement a helper function `dayOfYear(year, month, day)`: how many days passed since beginning of the year till the given day.

Here are two example test cases:

```
    elapsedDays(2021, 1, 1, 2021, 1, 1)  # should be 0
    elapsedDays(2021, 1, 1, 2021, 1, 2)  # should be 1
```

Question 4: [24p] Data manipulation and visualization.

Libraries

```
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
```

Input table

You will analyse the `apidae` (bees species) table (a simplified version of a publicly available dataset) of observed bee species in the Netherlands. The table is provided in the file `apidae.csv`. Each row represents a single observation from a certain date `eventDate`. The observations are included from different sources, such as a preserved specimen or individual observations.

The following columns are available in `apidae`:

- `family`: Taxonomic rank between `order` and genus (`order` taxonomic rank not included in the table)
- `genus`: Taxonomic rank between `family` and `species`
- `species`: Basic taxonomic rank
- `stateProvince`: Provinces of the Netherlands where the observations were recorded.
- `individualCount`: Count of observation (how many bees were observed)
- `occurrenceID`: Reference to the source of the observation.
- `decimalLatitude`: Geographic coordinate system (north-south)
- `decimalLongitude`: Geographic coordinate system (east-west)
- `eventDate`: Observation date

Reading data

Use the following code to read the `apidae` dataset as pandas DataFrame with `eventDate` correctly imported as `datetime` object:

```
apidae = pd.read_csv('apidae.csv', parse_dates=['eventDate'])
```

4.A. [3p] Table Summary

Produce **exactly** the summary output shown below from apidae DataFrame. It contains the number of distinct values in each variable (counting nan, if present, as a separate category). The (*) marks variables with no missing values.

```
family                1 (*)
genus                13
species              90
stateProvince        22
individualCount      93
occurrenceID      70264
decimalLatitude     989
decimalLongitude   1047
eventDate         11749 (*)
```

4.B. [3p] Query: Number of distinct species per province

Produce the counts of the numbers of different species per province. Sort the output by the number of species in descending order.

```
stateProvince
Limburg                      82
Gelderland                   72
Noord-Brabant                67
…
```

4.C. [3p] Query: Which species is the most common in Noord-Holland?

In Noord-Holland, for which species is the sum of individual counts the highest? Show the name of the species and the total number of individual counts observed.

4.D. [3p] Query: Percentages

Calculate and show the percentages of numbers of individuals counted per province. The sum of all percentages should be equal to 100%. Sort the output by the percentage in descending order. Do not include rows with missing values in the `stateProvince` or `individualCount` columns.

```
stateProvince
Zuid-Holland                  22.556752
Utrecht                       12.612995
Gelderland                    11.833657
…
```

4.E. [4p] Cross tabulation

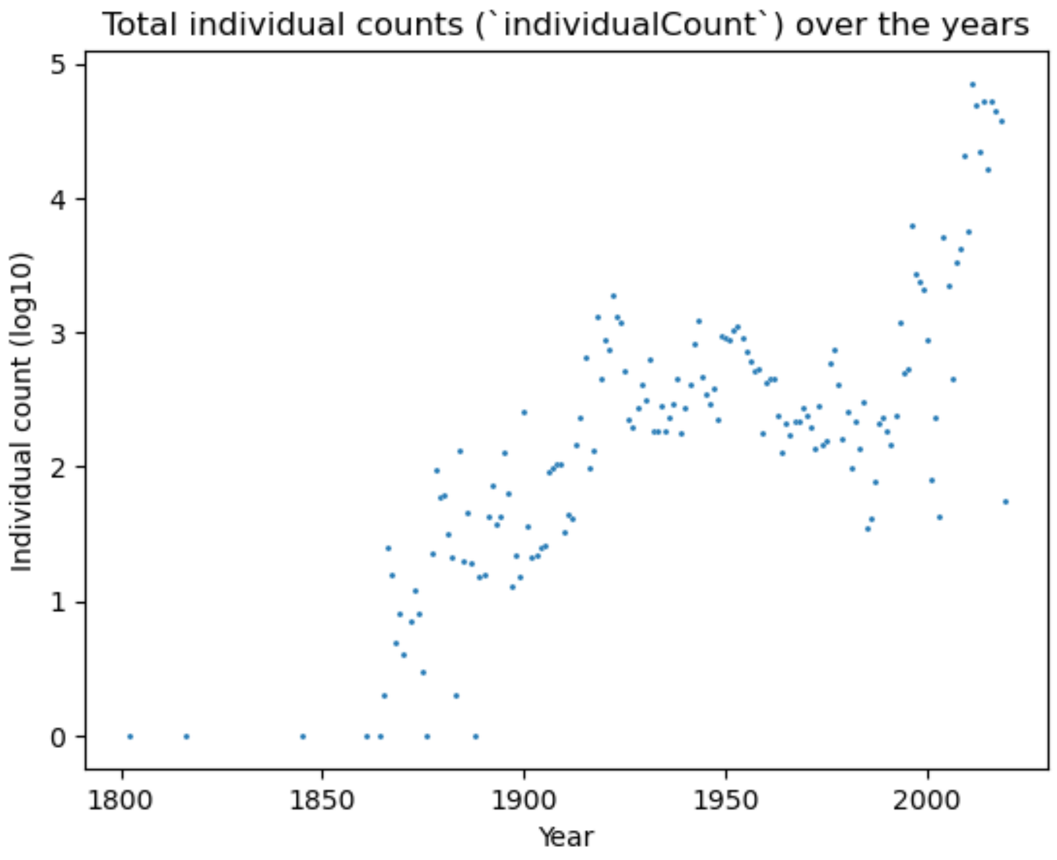
For observations on genus `Anthophora` produce a cross-tabulation of the total number of individuals per species (in columns) and per provinces (in rows). Fill the missing values with zeros.

```
species        Anthophora bimaculata  Anthophora furcata  …
stateProvince
Drenthe                          0.0                 4.0  …
Flevoland                        0.0                 0.0  …
Friesland                        0.0                 1.0  …
Gelderland                       3.0                17.0  …
…
```

4.F. [4p] Scatterplot

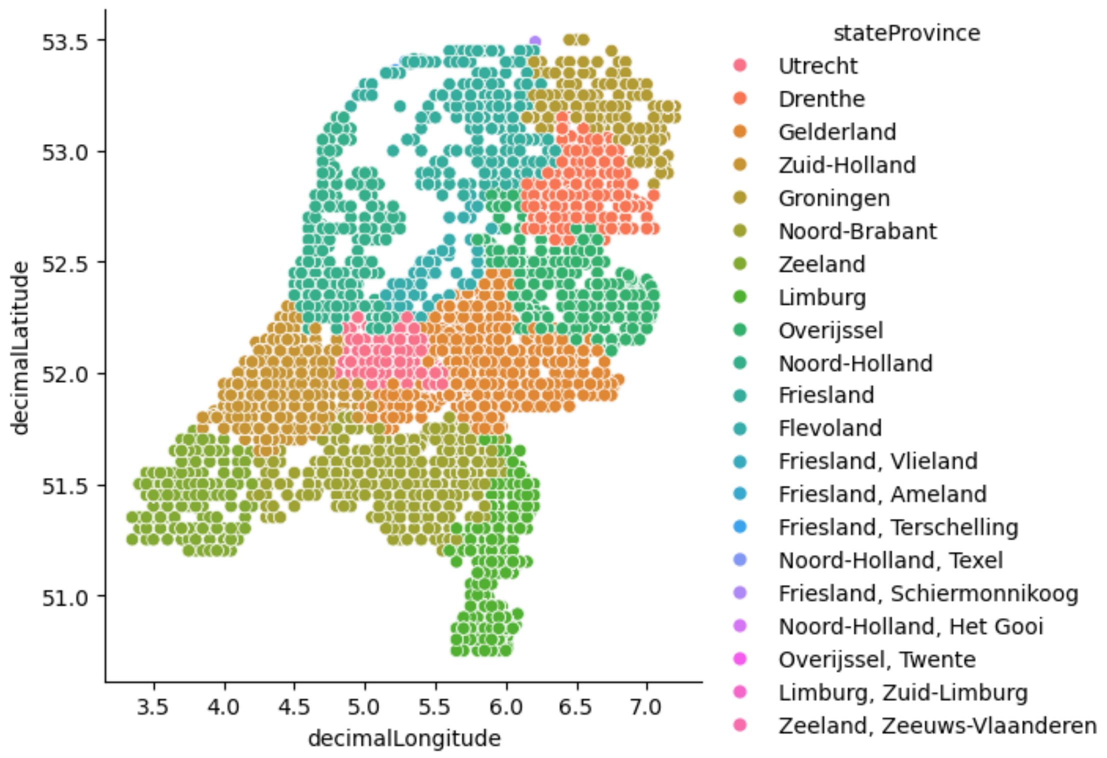
Reproduce the following scatter plot. It does not need to be identical, but it should contain the same information. The plot and axes titles must be set to similar texts. The numbers shown on the vertical scale should be after the log10 transformation. On the horizontal scale grouping is done on years (months/days are ignored). Colors and point sizes/shapes are not important.

Hint: panda.Series.dt.year



4.G. [4p] Geolocation scatterplot

- [2p] Reproduce the following scatter plot. It does not need to be identical, but it should contain the same information. Provinces should have different colors, and there should be a legend mapping each color to its province name.
- [2p] Connect with a straight line the most eastern to the most western observation. Do the same for the most northern to the most southern. The extreme points may not be unique, in that case select one at random.


decimalLatitude
decimalLongitude
stateProvince
Utrecht
Drenthe
Gelderland
Zuid-Holland
Groningen
Noord-Brabant
Zeeland
Limburg
Overijssel
Noord-Holland
Friesland
Flevoland
Friesland, Vlieland
Friesland, Ameland
Friesland, Terschelling
Noord-Holland, Texel
Friesland, Schiermonnikoog
Noord-Holland, Het Gooi
Overijssel, Twente
Limburg, Zuid-Limburg
Zeeland, Zeeuws-Vlaanderen