Assignment 1

Supplementary Subject in Cultural Data Science, Fall 2025

Réka Forgó

5th Semester, Cognitive Science Bsc.

Faculty of Arts, Aarhus University,

Jens Chr. Skous Vej 2, 8000 Aarhus, Denmark

26/09/2025

Table of Contents:

Link to my Github:

https://github.com/RekaForgo/CULTURALDATASCIENCE_CODE

week2

September 23, 2025

1 Week 2 - Variables

Instructor: Johanne Sejrskild
Date: 10.08.2025

Student: Réka Forgó (au737257@uni.au.dk)

Date of latest changes: 22.09.2025

Good afternoon
Today we are going to work with variables and text as data. The green excercises will be highly linked to what you did yesterday with Anna. If you find them challenging use Annas powerpoint as a help or ask. If you want to challenge yourself, try and do them all without using any help. In the yellow and red excercises we will be working with text as data. Here we will work with litterary classics and exploring the language used.

Structure of the notebook:
Green excercises

Variables and storing data in them

Container variable types

Accessing elements

Quiz

Yellow excercises

Build a lexicon of a word in a book of you choosing

Compare lengths of the books

Red excercises

Find the most frequent words

Compare pronouns used in books written by male and female authors

Start with the first excercise, and then continue in order. Feel free to work together, and see how far you can get.
The important thing is to learn, not to solve all the challenges!

Green excercises

1.1 Variables and storing data in them

Fill in the blanks:

Which variable does these data types belong to?
Whole numbers = int Decimal numbers = float Collections of characters = str
True or False values = bool

[46]:
```
# Add some values to the variables below and test if you are correct.
# you can use the type() function to check the data type of a variable.

x = 10
name = "John"
this_is_fun = True
height = 5.9

print(type(x))              # should be int
print(type(name))           # should be str
print(type(this_is_fun))    # should be bool
print(type(height))         # should be float
```

```
<class 'int'>
<class 'str'>
<class 'bool'>
<class 'float'>
```

1.2 Container variable types

Python provides several built-in container types, including lists, dictionaries, tuples, and sets. These containers can store collections of data and are essential for effective data manipulation.

Lists are ordered collections of items that can be changed (mutable). Lists are defined by square brackets [].
Dictionaries store key-value pairs, are unordered, and mutable. Defined by curly braces {}.
Tuples are ordered collections of items that cannot be changed (immutable). Defined by parentheses ().
Sets are unordered collections of unique items, mutable but items must be immutable. Defined by curly braces {} or the set() function.

[47]:
```
# create a list, dictionary, tuple and a set.

# list
my_list = [1, 2, 3, 4, 5, 44, 345, 2345, 234, 234, 23]
print(type(my_list))

# dictionary
my_dict = {"name": "Reka", "age": 22}
print(type(my_dict))
```

```
# tuple
my_tuple = (1, 2, 3, 4, 10)
print(type(my_tuple))

# set
my_set = {1, 2, 3}
print(type(my_set))
```

```
<class 'list'>
<class 'dict'>
<class 'tuple'>
<class 'set'>
```

1.3 Acessing elements

Access a value from each container variable you just created

[48]:
```
# list - Access elements by index, starting at 0.

print(my_list[0])  # first element
print(my_list[6])  # seventh element

# dictionary - Access values by their key.
print(my_dict["name"])  # value for key "name"
print(my_dict["age"])   # value for key "age"

# tuple - Access elements by index, just like lists.
print(my_tuple[0])  # first element
print(my_tuple[3])  # fourth element

# set - Cannot access items by index because sets are unordered. but you can␣
 ↪check if an item is in the set by using the 'in' keyword.
print(1 in my_set)  # True
print(10 in my_set)  # False
```

```
1
345
Reka
22
1
4
True
False
```

1.4 Quiz

1. Which of the following correctly adds a new element “orange” to the fruits list?

fruits.add(orange)
fruits.append("orange")
fruits.insert{"orange"}
fruits += ["orange"]

Answer: You can use fruits.append("orange") or fruits += ["orange]

2. How do you add a new key-value pair "gender": "female" to the person dictionary?

person.add("gender" = "female")
person["gender"] = "female"
person.append("gender": "female")
person.insert("gender" = "female")

Answer: person["gender"] = "female" to ad. a new key-value pair.

3. Which of the following statements about tuples is true?

Tuples are mutable.
You can add new elements to a tuple.
You access tuple elements by key.
Tuples are ordered and immutable.

Answer: Tuples are ordered and immutable is true.

4. What will be the result of adding an existing element to a set?

The set will add another instance of the element.
The set will remain unchanged because it only holds unique elements.
The operation will result in an error.
The set will reorder its elements.

Answer: The set will remain unchanged because it only holds unique elements.

Yellow excercises

Before you start on the *yellow excercises* you have to install and import the package containing all the litterary classics.
You will also get an example of how to build a lexicon of the use of the word "whale" in Moby Dick and the words surrounding it.

So:
Run each cell underneath one at a time, in order. If something in one cell doesn't work right, it might be because you have overwritten a variable, so try going back and running all the previous cells again.

[49]:
```
# install the natural language toolkit package (nltk), which has a copy of the␣
↪books.

%pip install nltk
```

```
Requirement already satisfied: nltk in /opt/anaconda3/lib/python3.12/site-
packages (3.9.1)
Requirement already satisfied: click in /opt/anaconda3/lib/python3.12/site-
```

```
packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /opt/anaconda3/lib/python3.12/site-
packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in
/opt/anaconda3/lib/python3.12/site-packages (from nltk) (2024.9.11)
Requirement already satisfied: tqdm in /opt/anaconda3/lib/python3.12/site-
packages (from nltk) (4.66.5)
Note: you may need to restart the kernel to use updated packages.
```

[50]:
```
# import the nltk package so that it is accessible to Python, and download a␣
↪collection of books from Project Gutenberg
import nltk
nltk.download('gutenberg')
```

```
[nltk_data] Downloading package gutenberg to
[nltk_data]     /Users/rekaforgo/nltk_data…
[nltk_data]   Package gutenberg is already up-to-date!
```

[50]:
```
True
```

Now we have installed and imported the books, but we want to extract one and do a bit of data cleeaning so we can look at the words used.
I have chosen to look at Moby Dick so lets find that

[51]:
```
# Create a variable called "mobydick" which contains the text of the book.
mobydick = nltk.corpus.gutenberg.raw('melville-moby_dick.txt')

# make all characters lowercase
mobydick = mobydick.lower()

# remove the "\n" and "\r" characters, which indicate line breaks in the text␣
↪(newlines)
mobydick = mobydick.replace('\n', ' ')
mobydick = mobydick.replace('\r', ' ')

# split up the text into a long list of individual words
mobydick = mobydick.split()
```

Question: Why is it necessary to split the text into a list of words in order to do analysis on them?

Answer: So we can treat them as tokens and do calculations based on their frequencies, etc.

[52]:
```
# make a variable called "concordance", and fill it with every occurrence of␣
↪the word "whale", and a few words preceeding and following "whale"
lexicon = []
for i, val in enumerate(mobydick):
    if val == "whale":
            lexicon.append(str(' '.join(mobydick[i-5:i+5])))
```

[53]:
```
# take a look at what the algorithm has found
print(lexicon[0:10])
```

```
['at least, take the higgledy-piggledy whale statements, however authentic, in',
'that monstrous bulk of the whale or ork we have', "paine, like as the wounded
whale to shore flies thro'", '--dryden\'s annus mirabilis. "while the whale is
floating at the', "jonas-in-the-whale. … some say the whale can't open his
mouth,", 'i was told of a whale taken near shetland, that', 'that he caught once
a whale in spitzbergen that was', 'and the breath of the whale is frequendy
attended with', 'contemptible in the comparison. the whale is doubtless the
largest', 'iceland in 1772. "the spermacetti whale found by the nantuckois,']
```

[54]:
```
# let's see how many instances of the word "whale" were found
len(lexicon)
```

[54]:
```
528
```

Let's try again, but this time let's just search for "the whale", not "whale" by itself. Try to comment each line of the code and explain what it does

[55]:
```
lexicon_twowords = []
for i, val in enumerate(mobydick): # Iterate through the text
    if val == "whale": # Check if the current word is "whale"
        if mobydick[i-1] == "the": # Check if the previous word is "the"
            lexicon_twowords.append(str(' '.join(mobydick[i-5:i+5]))) # If both␣
↪conditions are met, append the context to lexicon_twowords
```

[56]:
```
# take a look at what the algorithm has found
print(lexicon_twowords[0:10])
```

```
['that monstrous bulk of the whale or ork we have', '--dryden\'s annus
mirabilis. "while the whale is floating at the', "jonas-in-the-whale. … some
say the whale can't open his mouth,", 'and the breath of the whale is frequendy
attended with', 'contemptible in the comparison. the whale is doubtless the
largest', 'of the shipwreck of the whale ship essex of nantucket,', 'english
miles. … "sometimes the whale shakes its tremendous tail', 'the known species
of the whale tribe." --frederick debell bennett\'s', 'while." --miriam coffin or
the whale fisherman. "the whale is', 'for your lives!\'" --wharton the whale
killer. "so be cheery,']
```

1.5 Build a lexicon of a word in a book of you choosing

Now, in the cell below, modify the code from above to search for a word in a book of your own choosing. Give your lexicon a meaningful name that is different from the ones above and comment your code so you understand what is happening each step of the way. But first lets have a look at the books you can choose from.

[57]:
```
nltk.corpus.gutenberg.fileids()
```

```
emma = nltk.corpus.gutenberg.raw('austen-emma.txt')
```

[58]:
```
# Add your code to build your own lexicon here, using the same method as above.
# make all characters lowercase
emma = emma.lower()

# remove the "\n" and "\r" characters, which indicate line breaks in the text␣
↪(newlines)
emma = emma.replace('\n', ' ')
emma = emma.replace('\r', ' ')

# split up the text into a long list of individual words
emma = emma.split()
```

[59]:
```
lexicon_emma = []
for i, val in enumerate(emma):
    if val == "emma":
            lexicon_emma.append(str(' '.join(emma[i-5:i+5])))
# take a look at what the algorithm has found
print(lexicon_emma[0:10])
```

```
['1816] volume i chapter i emma woodhouse, handsome, clever, and', 'friend very
mutually attached, and emma doing just what she', 'of this beloved friend that
emma first sat in mournful', 'a mile from them; but emma was aware that great',
'was a melancholy change; and emma could not but sigh', 'of her life at
hartfield. emma smiled and chatted as', 'her how we all are." emma spared no
exertions to', 'a fanciful, troublesome creature!" said emma playfully. "that is
what', 'who could see faults in emma woodhouse, and the only', 'was not
particularly agreeable to emma herself, she knew it']
```

1.6 Compare lengths of books

We can use the command `len` to find how many items there are in a list. E.g., to find the number of words in the list called `mobydick`, from earlier, we can write: `len(mobydick)`.

Use the starter code below to find out which book in the books included in `nltk` has the most words.

[60]:
```
# One way to do it: Print all the titles and numbers of words
# starter code:

books = nltk.corpus.gutenberg.fileids()

for title in books:
    book = nltk.corpus.gutenberg.raw(title)
    book = book.split()
    print(f"{title}: {len(book)} words")
```

```
austen-emma.txt: 158167 words
austen-persuasion.txt: 83308 words
austen-sense.txt: 118675 words
bible-kjv.txt: 821133 words
blake-poems.txt: 6845 words
bryant-stories.txt: 45988 words
burgess-busterbrown.txt: 15870 words
carroll-alice.txt: 26443 words
chesterton-ball.txt: 81598 words
chesterton-brown.txt: 71626 words
chesterton-thursday.txt: 57955 words
edgeworth-parents.txt: 166070 words
melville-moby_dick.txt: 212030 words
milton-paradise.txt: 79659 words
shakespeare-caesar.txt: 20459 words
shakespeare-hamlet.txt: 29605 words
shakespeare-macbeth.txt: 17741 words
whitman-leaves.txt: 122070 words
```

Optional more advanced way to do it, for those with python experience up for a challenge

[61]:
```
# Another way to do it: Make a list of titles and a list of wordcounts, put it␣
 ↪in a dataframe, then sort them based on wordcount
%pip install pandas
import pandas as pd


# starter code:
books = nltk.corpus.gutenberg.fileids()

titles = []
numwords = []
for title in books:
    book = nltk.corpus.gutenberg.raw(title)
    book = book.split()
    titles.append(title)
    numwords.append(len(book))
df = pd.DataFrame({'title': titles, 'numwords': numwords})
df = df.sort_values(by='numwords', ascending=False)
df
```

```
Requirement already satisfied: pandas in /opt/anaconda3/lib/python3.12/site-
packages (2.2.2)
Requirement already satisfied: numpy>=1.26.0 in
/opt/anaconda3/lib/python3.12/site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/anaconda3/lib/python3.12/site-packages (from pandas) (2.9.0.post0)
```

```
Requirement already satisfied: pytz>=2020.1 in
/opt/anaconda3/lib/python3.12/site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in
/opt/anaconda3/lib/python3.12/site-packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.12/site-
packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

[61]:
```
                        title  numwords
3               bible-kjv.txt    821133
12      melville-moby_dick.txt    212030
11      edgeworth-parents.txt    166070
0             austen-emma.txt    158167
17         whitman-leaves.txt    122070
2            austen-sense.txt    118675
1       austen-persuasion.txt     83308
8          chesterton-ball.txt     81598
13        milton-paradise.txt     79659
9         chesterton-brown.txt     71626
10     chesterton-thursday.txt     57955
5          bryant-stories.txt     45988
15     shakespeare-hamlet.txt     29605
7            carroll-alice.txt     26443
14     shakespeare-caesar.txt     20459
16    shakespeare-macbeth.txt     17741
6     burgess-busterbrown.txt     15870
4             blake-poems.txt      6845
```

Red excercises

1.7 Find the most frequent words in the book you choose earlier

`nltk` has a built-in function called `FreqDist` which counts up how many times each word in a text occurs. So, if you have a list called `words` which contains all the words in a book, you can find the frequencies of all of them by writing `freq = nltk.FreqDist(words)`. You can then get the ten most common words by writing `freq.most_common(10)` and so on.
What are the ten most common words in your book?

[62]:
```python
# starter code:

book = nltk.corpus.gutenberg.raw('austen-emma.txt')
words = book.lower()
words = words.replace('\n', ' ')
words = words.replace('\r', ' ')
words = words.split()

# Most frequent words
freq = nltk.FreqDist(words)
```

```
freq.most_common(10)
```

[62]:
```
[('the', 5120),
 ('to', 5079),
 ('and', 4445),
 ('of', 4196),
 ('a', 3055),
 ('i', 2602),
 ('was', 2302),
 ('she', 2169),
 ('in', 2091),
 ('not', 2028)]
```

You might not feel like the ten most frequent words are of any real value in a potential project for analysis or comparison. This is due to the fact that the most frequent words in our language is what we call *stopwords*. These words like "a" and "the" are so common in English, that they don't really tell us much about the text.

That is why we often remove "stopwords", that is, a list of the most common words in English, before e.g. counting frequencies.

There are several of these lists available, in English as well as other languages, such as Danish.

Below is some starter code to remove stopwords. Use these snippets to see what the most common words in your book is after removing these most common words.

[63]:
```
# list of stopwords

stopwords = ["", "i", "me", "my", "myself", "we", "our", "ours", "ourselves",␣
 ↪"you", "your", "yours", "yourself", "yourselves", "he", "him", "his",␣
 ↪"himself", "she", "her", "hers", "herself", "it", "its", "itself", "they",␣
 ↪"them", "their", "theirs", "themselves", "what", "which", "who", "whom",␣
 ↪"this", "that", "these", "those", "am", "is", "are", "was", "were", "be",␣
 ↪"been", "being", "have", "has", "had", "having", "do", "does", "did",␣
 ↪"doing", "a", "an", "the", "and", "but", "if", "or", "because", "as",␣
 ↪"until", "while", "of", "at", "by", "for", "with", "about", "against",␣
 ↪"between", "into", "through", "during", "before", "after", "above", "below",␣
 ↪"to", "from", "up", "down", "in", "out", "on", "off", "over", "under",␣
 ↪"again", "further", "then", "once", "here", "there", "when", "where", "why",␣
 ↪"how", "all", "any", "both", "each", "few", "more", "most", "other", "some",␣
 ↪"such", "no", "nor", "not", "only", "own", "same", "so", "than", "too",␣
 ↪"very", "s", "t", "can", "will", "just", "don", "should", "now"]

# code to remove stopwords.
words = [word for word in words if word not in stopwords]
```

[64]:
```
# Most frequent words after removing stopwords
freq = nltk.FreqDist(words)
freq.most_common(10)
```

```
[64]: [('mr.', 1097),
       ('could', 800),
       ('would', 795),
       ('mrs.', 675),
       ('miss', 568),
       ('must', 543),
       ('emma', 481),
       ('much', 427),
       ('every', 425),
       ('said', 392)]
```

1.8 Compare the pronouns used in books written by male and female authors

This could be an example of a project
One could imagine that authors draw most inspiration from there own life and therefore write about characters resembeling themselves.
Investigate whether male and femate authors of novels write more pronouns they identify with themselves.

Is there any evidence for this postulation in our data?

If you can, it would be neat to make a loop that goes through all books, and creates a dataframe that you afterwards can sort in to remove non-novels :)

```
[65]: # Example of structure:

# Make a set of male and female pronouns (2 sets)


male_pronouns = {"he", "him", "his", "himself"}
female_pronouns = {"she", "her", "hers", "herself"}

# Make a dictionary of all titles in the corpus and the authors gender (1␣
 ↪dictionary)

title_gender = {
    "austen-emma.txt": "female",
    "austen-persuasion.txt": "female",
    "austen-sense.txt": "female",
    "bible-kjv.txt": "male",
    "blake-poems.txt": "male",
    "bryant-stories.txt": "male",
    "burgess-busterbrown.txt": "male",
    "carroll-alice.txt": "male",
    "chesterton-ball.txt": "male",
    "chesterton-brown.txt": "male",
    "chesterton-thursday.txt": "male",
    "edgeworth-parents.txt": "female",
    "melville-moby_dick.txt": "male",
    "milton-paradise.txt": "male",
    "shakespeare-caesar.txt": "male",
    "shakespeare-hamlet.txt": "male",
    "shakespeare-macbeth.txt": "male",
    "whitman-leaves.txt": "male",
}



titles, genders, num_female_pronouns, num_male_pronouns = [], [], [], []

for title, gender in title_gender.items():
    book = nltk.corpus.gutenberg.raw(title)
    words = book.lower().split()
    male_count = sum(word in male_pronouns for word in words)
    female_count = sum(word in female_pronouns for word in words)

    titles.append(title)
    genders.append(gender)
    num_female_pronouns.append(female_count)
    num_male_pronouns.append(male_count)

df = pd.DataFrame({
    "title": titles,
    "gender": genders,
    "num_female_pronouns": num_female_pronouns,
    "num_male_pronouns": num_male_pronouns,
})

print(df.sort_values("title"))
```

```
                      title  gender  num_female_pronouns  num_male_pronouns
0            austen-emma.txt  female                 4292               3284
1      austen-persuasion.txt  female                 2223               1884
2           austen-sense.txt  female                 3864               2477
3              bible-kjv.txt    male                 2530              22210
4            blake-poems.txt    male                   48                117
5         bryant-stories.txt    male                  660               1741
6    burgess-busterbrown.txt    male                    3                911
7          carroll-alice.txt    male                  769                233
8        chesterton-ball.txt    male                  190               2509
9       chesterton-brown.txt    male                  219               2493
10  chesterton-thursday.txt    male                   15               2115
11     edgeworth-parents.txt  female                 2606               4814
12    melville-moby_dick.txt    male                  396               4856
13       milton-paradise.txt    male                  412               1978
14    shakespeare-caesar.txt    male                   14                426
```

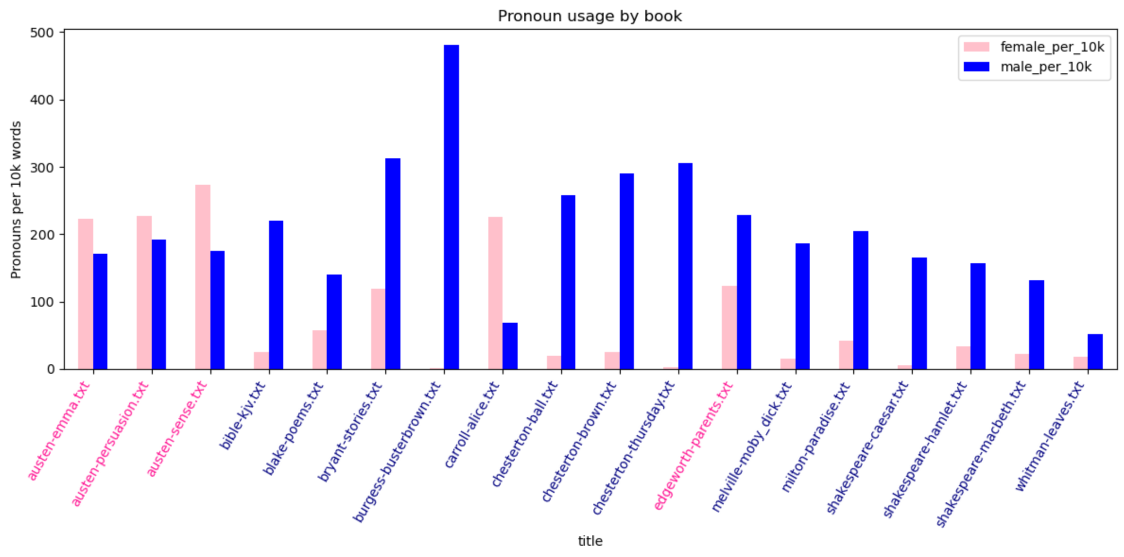
```
15   shakespeare-hamlet.txt    male                  125                 588
16  shakespeare-macbeth.txt    male                   53                 306
17        whitman-leaves.txt    male                  278                 809
```

[66]:
```
#remove non-novels
non_novels = [
    "bible-kjv.txt",
    "blake-poems.txt",
    "whitman-leaves.txt",
    "milton-paradise.txt",
    "shakespeare-caesar.txt",
    "shakespeare-hamlet.txt",
    "shakespeare-macbeth.txt",
    "chesterton-ball.txt",
    "chesterton-brown.txt",
    "chesterton-thursday.txt",
    "burgess-busterbrown.txt",   # children's stories
    "bryant-stories.txt"         # children's stories
]

df_novels = df[~df["title"].isin(non_novels)].copy()

print("\nTitles remaining after filter:")
print(df_novels["title"].unique())
```

```
Titles remaining after filter:
['austen-emma.txt' 'austen-persuasion.txt' 'austen-sense.txt'
 'carroll-alice.txt' 'edgeworth-parents.txt' 'melville-moby_dick.txt']
```

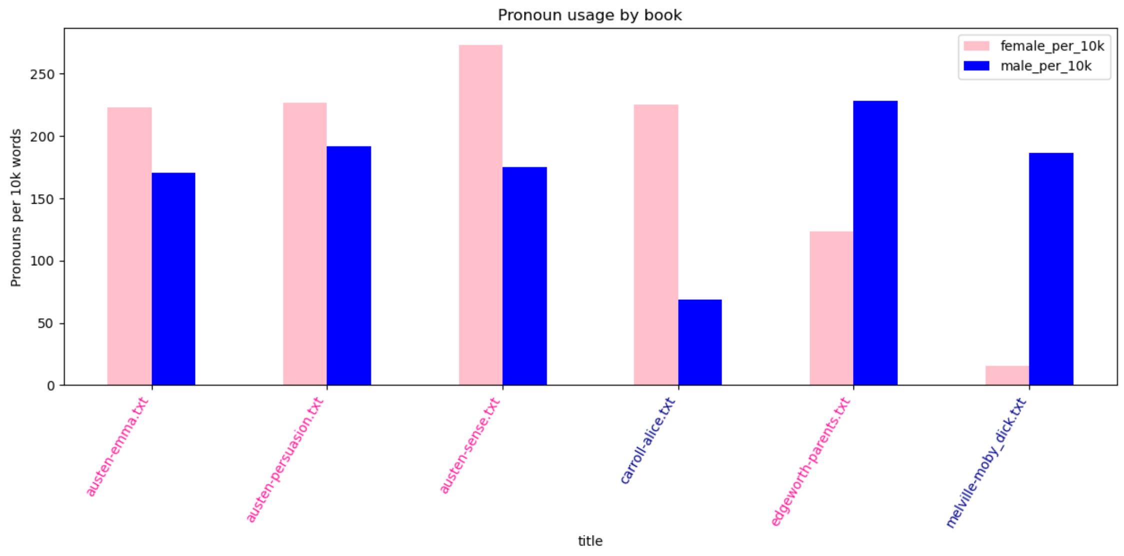
[67]:
```
#visualise this data - Also: normalise the distribution, because the bible is␣
 ↪much longer - higher overall pronoun count
import matplotlib.pyplot as plt
# Add a column with total number of tokens (words) in each book
df["tokens"] = df["title"].apply(lambda t: len(nltk.corpus.gutenberg.words(t)))

# Example: use normalized counts per 10k words
df["male_per_10k"] = df["num_male_pronouns"] / df["tokens"] * 10000
df["female_per_10k"] = df["num_female_pronouns"] / df["tokens"] * 10000

# Plot as grouped bars
ax = df.plot(
    kind="bar",
    x="title",
    y=["female_per_10k", "male_per_10k"],
    color=["pink", "blue"],
    figsize=(12, 6)
)
```

```
plt.ylabel("Pronouns per 10k words")
plt.title("Pronoun usage by book")

# --- color the tick labels by author gender ---
title_to_gender = dict(zip(df["title"], df["gender"]))
for label in ax.get_xticklabels():
    title = label.get_text()
    if title_to_gender[title] == "female":
        label.set_color("deeppink")
    else:
        label.set_color("navy")

plt.xticks(rotation=60, ha="right")
plt.tight_layout()
plt.show()
```



[68]:
```
#now for novels

# Add a column with total number of tokens (words) in each book
df_novels["tokens"] = df_novels["title"].apply(lambda t: len(nltk.corpus.
↪gutenberg.words(t)))

# Example: use normalized counts per 10k words
df_novels["male_per_10k"] = df_novels["num_male_pronouns"] /␣
↪df_novels["tokens"] * 10000
df_novels["female_per_10k"] = df_novels["num_female_pronouns"] /␣
↪df_novels["tokens"] * 10000
```

```python
# Plot as grouped bars
ax = df_novels.plot(
    kind="bar",
    x="title",
    y=["female_per_10k", "male_per_10k"],
    color=["pink", "blue"],
    figsize=(12, 6)
)

plt.ylabel("Pronouns per 10k words")
plt.title("Pronoun usage by book")

# --- color the tick labels by author gender ---
title_to_gender = dict(zip(df_novels["title"], df_novels["gender"]))


for label in ax.get_xticklabels():
    title = label.get_text()
    if title_to_gender[title] == "female":
        label.set_color("deeppink")
    else:
        label.set_color("navy")

plt.xticks(rotation=60, ha="right")
plt.tight_layout()
plt.show()
```



Discuss:

What is your findings and what is the limitations of this project

This project is limited by multiple things, one being only a small set of specific authors and types of literature (novels), which limits the generalisability of findings. Another limitation is that the frequency of pronoun use does not account for contextual information, so it might not directly infer bias by the author.

week3

September 23, 2025

1 Week 3 - Loops, Conditions and Functions

Instructor: Johanne Sejrskild
Date: 12.08.2025

Student: Réka Forgó (au737257@uni.au.dk)

Date of latest changes: 23.09.2025

Good afternoon
Today we are going to work with loops, condions and functions. The green excercises will be highly linked to what you livecoded with Anna. If you find them challenging use yesterdays work as a help or ask. If you want to challenge yourself, try and do them all without using any help. In the yellow excercises we will make a password checker and a function that converts temperatures. And for the red excercise we will make a function that calculates grades

Structure of the notebook:
Green excercises

Conditional Statements

Loops

Functions

Yellow excercises

Password checker

Temperature converter

Red excercises

Grade calculator

Optional extra: Grade calculator as a class

Start with the first excercise, and then continue in order. Feel free to work together, and see how far you can get.
The important thing is to learn, not to solve all the challenges!

---

[2]:
```
# Before we start we need to import the necessary packages
%pip install pandas
%pip install lxml
import pandas as pd
```

```
Requirement already satisfied: pandas in /opt/anaconda3/lib/python3.12/site-
packages (2.2.2)
Requirement already satisfied: numpy>=1.26.0 in
/opt/anaconda3/lib/python3.12/site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/anaconda3/lib/python3.12/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/opt/anaconda3/lib/python3.12/site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in
/opt/anaconda3/lib/python3.12/site-packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.12/site-
packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: lxml in /opt/anaconda3/lib/python3.12/site-
packages (5.2.1)
Note: you may need to restart the kernel to use updated packages.
```

Green excercises

1.1 Conditional Statements

The *if* statement

[3]:
```
# Basic syntax
# if condition:
    # code to execute if condition is True
```

[4]:
```
# Example of if statement
# Before running: What will happen and why?

x = 10
if x > 5:
    print("x is greater than 5")
```

```
x is greater than 5
```

[5]:
```
# Exceise: Modify the code to check if x is less than 5 and print a different␣
 ↪message.

if x < 5:
    print("x is less than 5")
```

The *else* statement

[6]:
```python
# Basic syntax

# if condition:
    # code if condition is True
# else:
    # code if condition is False
```

[7]:
```python
# Example of if-else statement
# Before running: What will happen and why?

x = 3
if x > 5:
    print("x is greater than 5")
else:
    print("x is not greater than 5")
```

```
x is not greater than 5
```

[8]:
```python
# Excercise - Change the code above so that it prints "x is greater than 5." by␣
 ↪only changing one number/symbol.
# How many solutions can you find? and is some of them better than others?

if x < 5:
    print("x is greater than 5")
else:
    print("x is not greater than 5")
```

```
x is greater than 5
```

The *elif* statement (for multiple conditions)

[9]:
```python
# Basic syntax

# if condition1:
    # code if condition1 is True
# elif condition2:
    # code if condition2 is True
# else:
    # code if neither condition is True

```

[10]:
```python
# Example of if-elif-else statement
# Before running: What will happen and why?

x = 5
if x > 5:
    print("x is greater than 5")
elif x == 5:
    print("x is equal to 5")
```

```
else:
    print("x is less than 5")
```

```
x is equal to 5
```

[11]:
```
# Exercise - What happens when the condition for if and elif are the same? Why?


if x > 5:
    print("x is greater than 5")
elif x > 5:
    print("x is equal to 5")
else:
    print("x is less than 5")

# The first condition is checked, and if it's true, the corresponding block is␣
↪executed.
# Hence if if and elif are the same, the elif block will never be executed.
```

```
x is less than 5
```

1.2 Loops

Loops in Python are used to execute a block of code repeatedly. We will look at two different types of loops; the *for loop*, and the *while loop*.
The for loop iterates over sequences, so it keeps executing until it has reached the end of the loop.
The while loop keeps repeating while a condition is true, and first ends when the condition becomes false.

[12]:
```
# For loop - Basic syntax
# for element in sequence:
    # code to execute for each element

# Example of for loop
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

```
apple
banana
cherry
```

[13]:
```
# Exercise - Modify the code to print the length of each fruit instead of the␣
↪fruit itself.

for fruit in fruits:
    print(len(fruit))
```

```
5
6
```

```
6
```

[14]:
```python
# While loop - Basic syntax
# while condition:
    # code to execute while condition is True

# Example of while loop
x = 5
while x > 0:
    print(x)
    x -= 1
```

```
5
4
3
2
1
```

[15]:
```python
# Excercise - Modify the code to count up from 1 to 5 instead of counting down␣
 ↪from 5 to 1.

x = 1
while x <= 5:
    print(x)
    x += 1
```

```
1
2
3
4
5
```

1.3 Functions

[16]:
```python
# Heres a calculator
def add_two_numbers(a, b):
    result = a + b
    print(f"{a} + {b} = {result}")
```

[17]:
```python
# Try it out with different numbers
add_two_numbers(3, 5)
add_two_numbers(10, 20)
add_two_numbers(-1, 1)
```

```
3 + 5 = 8
10 + 20 = 30
-1 + 1 = 0
```

[18]:
```
# Make a subtraction function
def subtract_two_numbers(a, b):
    result = a - b
    print(f"{a} - {b} = {result}")

subtract_two_numbers(10, 5)
subtract_two_numbers(20, 30)
```

```
10 - 5 = 5
20 - 30 = -10
```

Yellow excercises

1.4 Password checker

You're making a simple password checker. How would you check if the password is correct?

Write code that asks for a password and checks if it follows the requirements: - It needs to be between 8 and 32 characters long

Make sure you print some meaningful message the user can understand. Try it with both the correct password and wrong passwords.

Hint: use the input function to allow for user inputs e.g. **input("Enter the password:")**

[19]:
```
# code here

def password_checker(password):
    if len(password) < 8:
        return "Password is too short"
    elif len(password) > 32:
        return "Password is too long"
    else:
        return "Password is valid"

input_password = input("Enter your password: ")
print(password_checker(input_password))
```

```
Password is too short
```

Now lets add some extra requirements: - The password needs to have at least 3 digits - And there needs to be an uppercase letter

[20]:
```
# Modified code here

def password_checker(password):
    if len(password) < 8:
        return "Password is too short"
    elif len(password) > 32:
        return "Password is too long"
    elif sum(char.isdigit() for char in password) < 3:
        return "Password must contain at least three digits"
    elif not any(char.isupper() for char in password):
        return "Password must contain at least one uppercase letter"
    else:
        return "Password is valid"

input_password = input("Enter your password: ")
print(password_checker(input_password))
```

```
Password must contain at least one uppercase letter
```

And the last bits, now we also need to ensure that the password contains: - 2 special characters

Make sure the user understands why a wrong password is rejected. And ensure that only a password that holds all requirements are passed.

[21]:
```
# Last modifications on the code - final password_checker

special = "!@#$%^&*()-_=+[]{}|;:'\",.<>?/`~" # Define special characters


def password_checker(password):
    if len(password) < 8:
        return "Password is too short"
    elif len(password) > 32:
        return "Password is too long"
    elif sum(char.isdigit() for char in password) < 3:
        return "Password must contain at least three digits"
    elif not any(char.isupper() for char in password):
        return "Password must contain at least one uppercase letter"
    elif sum(char in special for char in password) < 2:
        return "Password must contain at least two special characters"
    else:
        return "Password is valid"

input_password = input("Enter your password: ")
print(password_checker(input_password))
```

```
Password must contain at least one uppercase letter
```

1.5 Temperature converter

Lets now make a function that can convert celcius to fahrenheit. Make an IPO with your group and discuss what hte input, the process and the outpus should be.

The function should print an informational message and return something usable.

Hint: F = (celsius * 9/5) +32

[22]:
```python
# celsius to fahrenheit converter

def celsius_to_fahrenheit(celsius):
    if celsius < -273.15:
        return "Temperature below -273.15°C is not possible."
    else:
        fahrenheit = (celsius * 9/5) + 32
        return f"{celsius}°C is equal to {fahrenheit}°F"

# Try it out with different numbers

print(celsius_to_fahrenheit(0))
print(celsius_to_fahrenheit(100))
print(celsius_to_fahrenheit(-300))
```

```
0°C is equal to 32.0°F
100°C is equal to 212.0°F
Temperature below -273.15°C is not possible.
```

Now make a function that does the opposite.
Hint: C = ((fahrenheit-32)/(9/5))

[23]:
```python
# Fahrenheit to Celsius converter

def fahrenheit_to_celsius(fahrenheit):
    if fahrenheit < -459.67:
        return "Temperature below -459.67°F is not possible."
    else:
        celsius = (fahrenheit - 32) * 5/9
        return f"{fahrenheit}°F is equal to {celsius}°C"

# Try it out with different numbers
print(fahrenheit_to_celsius(32))
print(fahrenheit_to_celsius(212))
print(fahrenheit_to_celsius(-500))
```

```
32°F is equal to 0.0°C
212°F is equal to 100.0°C
Temperature below -459.67°F is not possible.
```

Well done! Now wouldn't it be neat if we could just wrap that into a function that we could call temperature_converter with two parameters "temperature" and "unit" so that we could convert both ways.

[26]:
```python
# Temperature converter here

def temperature_converter(value, scale):
    if scale == 'c': #check if the scale of the value is celsius
        if value < -273.15:
            return "Temperature below -273.15°C is not possible."
        else:
            fahrenheit = (value * 9/5) + 32
            return f"{value}°C is equal to {fahrenheit}°F"
    elif scale == 'f': #check if the scale of the value is fahrenheit
        if value < -459.67:
            return "Temperature below -459.67°F is not possible."
        else:
            celsius = (value - 32) * 5/9
            return f"{value}°F is equal to {celsius}°C"
    else:
        return "Invalid scale. Please use 'c' for Celsius or 'f' for Fahrenheit.
 ↪"

#Let's make input fields which asks the temperature value in one scale and␣
 ↪converts it to the other.
value = float(input("Enter the temperature value: "))
scale = input("Enter the scale (c for Celsius, f for Fahrenheit): ")
print(temperature_converter(value, scale))
```

```
34.0°F is equal to 1.1111111111111112°C
```

OBS: To ensure you can go back in 3 months time and read you code and understand the logics behind it it needs to be well commented.
So, take look at the yellow excercises and see if you need to add some meaningful comments about the logics and coding choices.

:))

Red excercises

1.6 Grade calculator

The scenario: You're helping a teacher who wants to calculate student grades quickly. The teacher has been doing these calculations by hand, which takes a long time and sometimes leads to mistakes.

The challenge: - Calculate the average of test scores - Convert percentage grades to letter grades (it's just easier than the danish grading metric ngl) - Calculate final grades with different weights for different assignments

Example data:
Tests count for 70%, homework counts for 30%

Test scores: 85, 93, 78, 96, 88 Homework average: 90

Now we start with one thing first. Each functions should do *one* thing effectively.

[27]:
```
# Make a function that calculate the average of test scores.
#we need to provide a list of scores to the function when we call it.
# we need to set it up so it is able to handle homework and test scores with␣
 ↪different weights
```

```
def calculate_average(test_scores, homework_scores, test_weight=0.7,␣
↪homework_weight=0.3):
    if not test_scores and not homework_scores:
        return "No scores provided."
    test_average = sum(test_scores) / len(test_scores) if test_scores else 0
    homework_average = sum(homework_scores) / len(homework_scores) if␣
↪homework_scores else 0
    overall_average = (test_average * test_weight) + (homework_average *␣
↪homework_weight)
    return print(f"Overall average score: {overall_average}"), overall_average

# Try it out with different scores
#print(calculate_average([85, 93, 78, 96, 88], [90]))

overall_average = calculate_average([85, 93, 78, 96, 88], [90])[1]
```

```
Overall average score: 88.6
```

Well done, now we move on to the next sub-task

Hint: Grade to lettter grade converter
90–100 → A
80–89 → B
70–79 → C
60–69 → D
0–59 → F

[28]:
```
# Make a function that converts the test scores to letter grades

def percentage_to_letter_grade(percentage):
    if percentage >= 90:
        return "A"
    elif percentage >= 80:
        return "B"
    elif percentage >= 70:
        return "C"
    elif percentage >= 60:
        return "D"
    else:
        return "F"



# Test with different percentages
print(percentage_to_letter_grade(overall_average))
```

```
B
```

Last task, We need to calculate the weighted grade

[29]:
```python
#def calculate_weighted_grade():

# already done above by the calculate_average function
```

Now we want to give the professor one function that uses all these three and wraps them in a neat little function with everything put together.

[30]:
```python
def calculate_student_grade(test_scores, homework_scores, test_weight=0.7,␣
 ↪homework_weight=0.3):
    if not test_scores and not homework_scores:
        return "No scores provided."
    overall_average = calculate_average(test_scores, homework_scores,␣
 ↪test_weight, homework_weight)[1] # just the numeric average (not the print␣
 ↪statement)
    return percentage_to_letter_grade(overall_average)

# Create input fields that ask for test scores and homework scores
input_test_scores = input("Enter test scores separated by commas: ")
input_homework_scores = input("Enter homework scores separated by commas: ")

# Fix the parsing to handle decimal numbers properly
test_scores = []
if input_test_scores.strip():
    for score in input_test_scores.split(","):
        try:
            test_scores.append(float(score.strip()))
        except ValueError:
            print(f"Warning: '{score.strip()}' is not a valid number, skipping..
 ↪.")

homework_scores = []
if input_homework_scores.strip():
    for score in input_homework_scores.split(","):
        try:
            homework_scores.append(float(score.strip()))
        except ValueError:
            print(f"Warning: '{score.strip()}' is not a valid number, skipping..
 ↪.")

print(f"Final Grade: {calculate_student_grade(test_scores, homework_scores)}")
```

```
Overall average score: 82.1
Final Grade: B
```

1.7 Optional red: Take your grade calculator and make it a class!

[31]:
```
# "Grader"

class Grader:
    def __init__(self, test_weight=0.7, homework_weight=0.3):
        """Initialize the grader with default weights"""
        self.test_weight = test_weight
        self.homework_weight = homework_weight

    def calculate_average(self, test_scores, homework_scores):
        """Calculate weighted average of test and homework scores"""
        if not test_scores and not homework_scores:
            return None, None, None

        test_average = sum(test_scores) / len(test_scores) if test_scores else 0
        homework_average = sum(homework_scores) / len(homework_scores) if␣
↪homework_scores else 0
        overall_average = (test_average * self.test_weight) + (homework_average␣
↪* self.homework_weight)
        return test_average, homework_average, overall_average

    def percentage_to_letter_grade(self, percentage):
        """Convert percentage to letter grade"""
        if percentage >= 90:
            return "A"
        elif percentage >= 80:
            return "B"
        elif percentage >= 70:
            return "C"
        elif percentage >= 60:
            return "D"
        else:
            return "F"

    def generate_grade_report(self, test_scores, homework_scores, student_name):
        """Generate a full grade report for a student"""
        if not test_scores and not homework_scores:
            return f"No scores provided for {student_name}"

        test_avg, hw_avg, overall_avg = self.calculate_average(test_scores,␣
↪homework_scores)
        letter_grade = self.percentage_to_letter_grade(overall_avg)

        return f"""
=== Grade Report for {student_name} ===
Weighting: Tests ({self.test_weight*100:.0f}%) | Homework ({self.
↪homework_weight*100:.0f}%)
Test Average: {test_avg:.1f}%
Homework Average: {hw_avg:.1f}%
Overall Average: {overall_avg:.1f}%
Final Grade: {letter_grade}
"""

grader = Grader()  # Uses default weights (70% tests, 30% homework)

test_scores = [85, 93, 78, 96, 88]
homework_scores = [90]

print(grader.generate_grade_report(test_scores, homework_scores, "Aelin␣
↪Galathynius"))
```

```
=== Grade Report for Aelin Galathynius ===
Weighting: Tests (70%) | Homework (30%)
Test Average: 88.0%
Homework Average: 90.0%
Overall Average: 88.6%
Final Grade: B
```

1.7.1 Wuhuu - now you are all done. Impressive!

If you have time alter the function above so it outputs a neat dtudent grade report with test scores, test average, homework average, final grade and letter grade.

week4_notebook

September 26, 2025

1 Week 4 - Data organising with Pandas

Author: Johanne Sejrskild
Date: 22.09.2025

Student: Réka Forgó (au737257@uni.au.dk)

Date of latest changes: 26.09.2025

Good afternoon
Today we are going to work with loops, condions and using ´pandas´to manipulate data. The green excercises will be highly linked to what you livecoded with Anna. If you find them challenging use yesterdays work as a help or ask. If you want to challenge yourself, try and do them all without using any help. In the yellow excercises we will do some data manipulation challenges using pandas. And we will skip the red tasks today as we have a lot on the program

Structure of the notebook:
Green excercises

Data wrangling on the iris dataset

Yellow excercises

Music sales challenge

Space mission challenge

Supervillan challenge

Start with the first excercise, and then continue in order. Feel free to work together, and see how far you can get.
The important thing is to learn, not to solve all the challenges!

---

[2]:
```
# Before we start we need to import the necessary packages
#%pip install pandas
#%pip install lxml
#%pip install scikit-learn

import pandas as pd
import requests # We might need this package to get some data from the web
```

```
from sklearn import datasets
```

Green excercises

1.1 Data organisation using a dataset about flowers

```
[3]: flower = datasets.load_iris()

     # convert to DataFrame
     df = pd.DataFrame(flower.data, columns=flower.feature_names)

     df.head()
```

```
[3]:    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
     0                5.1               3.5                1.4               0.2
     1                4.9               3.0                1.4               0.2
     2                4.7               3.2                1.3               0.2
     3                4.6               3.1                1.5               0.2
     4                5.0               3.6                1.4               0.2
```

Lets take a look at the data frame

```
[4]: # There are some commands in the library pandas that can give you a quick␣
       ↪overview of the data :)

     df.head()      # first 5 rows, if you put a number into the paranthesis you can␣
       ↪decide how many rows
     df.tail()      # last 5 rows
     df.info()      # summary of columns and types
     df.describe()  # quick statistics (for numbers)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
dtypes: float64(4)
memory usage: 4.8 KB
```

```
[4]:        sepal length (cm)  sepal width (cm)  petal length (cm)  \
     count         150.000000        150.000000         150.000000
     mean            5.843333          3.057333           3.758000
     std             0.828066          0.435866           1.765298
     min             4.300000          2.000000           1.000000
```

```
25%                5.100000          2.800000           1.600000
50%                5.800000          3.000000           4.350000
75%                6.400000          3.300000           5.100000
max                7.900000          4.400000           6.900000

       petal width (cm)
count        150.000000
mean           1.199333
std            0.762238
min            0.100000
25%            0.300000
50%            1.300000
75%            1.800000
max            2.500000
```

Selecting columns and rows

Try to run the cell below and figure out which output is linked to the code

[5]:
```
# If you want to select a specific column you can select it using the name:
print(df['sepal length (cm)'].head())

# If you would like to print one row, you can use the index of the row:
print(df.iloc[0])

# if you want to select a few rows of only a few columns you can also use␣
 ↪indexing:
print(df.iloc[0:3 , 0:2])  # first three rows, first two columns

# And if you want to select specific data, you can specify a single row and␣
 ↪column:
print(df.iloc[2,0])  # second row, first column

# Or use the column name:
print(df.loc[2, "sepal length (cm)"]  )
```

```
0    5.1
1    4.9
2    4.7
3    4.6
4    5.0
Name: sepal length (cm), dtype: float64
sepal length (cm)    5.1
sepal width (cm)     3.5
petal length (cm)    1.4
petal width (cm)     0.2
Name: 0, dtype: float64
   sepal length (cm)  sepal width (cm)
0                5.1               3.5
```

```
1                 4.9               3.0
2                 4.7               3.2
4.7
4.7
```

Subsetting data
Subsetting is the process of retrieving just the parts of large files which are of interest for a specific purpose.
This will come in handy for your projects when you have to work with potentially large data files

[6]:
```
# Let's try to select some data using conditionals

# Here we select all rows where the sepal length is larger than or equal to 5 cm
lengt_above_five = df[df["sepal length (cm)"] >= 5]
lengt_above_five.head()
```

[6]:
```
    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                 5.1               3.5                1.4               0.2
4                 5.0               3.6                1.4               0.2
5                 5.4               3.9                1.7               0.4
7                 5.0               3.4                1.5               0.2
10                5.4               3.7                1.5               0.2
```

[7]:
```
# Here we select all rows where the sepal length is larger than or equal to 5␣
 ↪cm and the sepal width is less than 2,5 cm
length_and_width = df[(df["sepal length (cm)"] >= 5) & (df["sepal width (cm)"]␣
 ↪< 3.5)]
length_and_width.head()
```

[7]:
```
    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
7                 5.0               3.4                1.5               0.2
20                5.4               3.4                1.7               0.2
23                5.1               3.3                1.7               0.5
25                5.0               3.0                1.6               0.2
26                5.0               3.4                1.6               0.4
```

[8]:
```
# Excercise - Find the longest petal length and the median petal length
# and subset the flowers that are between the median and one centimeter shorter␣
 ↪than max length

max_length = df["petal length (cm)"].max()
median_length = df["petal length (cm)"].median()
subset = df[(df["petal length (cm)"] > median_length) & (df["petal length␣
 ↪(cm)"] < max_length - 1)]
print(f"Max length: {max_length}, Median length: {median_length}")
print(subset.head())
```

```
Max length: 6.9, Median length: 4.35
    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
```

```
50                 7.0                3.2                 4.7                1.4
51                 6.4                3.2                 4.5                1.5
52                 6.9                3.1                 4.9                1.5
54                 6.5                2.8                 4.6                1.5
55                 5.7                2.8                 4.5                1.3
```

Sorting data
We can choose to sort our data in order of something of interest.

```
[9]: # we could sort the data by a specific column in both ascending and descending␣
       ↪order
     df_sorted = df.sort_values(by="sepal length (cm)", ascending=False) # change␣
       ↪direction by True/False so if you want ascending order set it to True
     df_sorted.head()
```

```
[9]:      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
     131                7.9               3.8                6.4               2.0
     135                7.7               3.0                6.1               2.3
     122                7.7               2.8                6.7               2.0
     117                7.7               3.8                6.7               2.2
     118                7.7               2.6                6.9               2.3
```

```
[10]: # Excercise - sort the data by petal width in ascending order and select the 10␣
        ↪flowers with the smallest petal width

      sorted_df = df.sort_values(by="petal width (cm)", ascending=True)
      subset_10 = sorted_df.head(10)

      print(subset_10)
```

```
    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
32                 5.2               4.1                1.5               0.1
13                 4.3               3.0                1.1               0.1
37                 4.9               3.6                1.4               0.1
9                  4.9               3.1                1.5               0.1
12                 4.8               3.0                1.4               0.1
0                  5.1               3.5                1.4               0.2
27                 5.2               3.5                1.5               0.2
28                 5.2               3.4                1.4               0.2
29                 4.7               3.2                1.6               0.2
30                 4.8               3.1                1.6               0.2
```

Flipping
Should you work with time seires data and would like to mirror (flip) your data, you can do this using pandas

```
[11]: print(df.head(5))
```

```
reversed_df = df.iloc[::-1]    # Flipping the dataframe horisontally (reverse␣
  ↪rows)

print(reversed_df.head(5))
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
149                5.9               3.0                5.1               1.8
148                6.2               3.4                5.4               2.3
147                6.5               3.0                5.2               2.0
146                6.3               2.5                5.0               1.9
145                6.7               3.0                5.2               2.3
```

Joining Sometimes we have multiple dataframes we woudl like to add together. Maybe you have been subsetting parts of an old dataframe to substract important information and would now like join them so you can begin your analysis.

```
[12]: # Joining a bit of the iris data with a new dataframe (we will make up some␣
        ↪data here)
      first_10 = df.iloc[0:10, :]  # selecting the first 10 rows of the iris data
      new_data = {"color": ["red", "blue", "green", "yellow", "purple", "red",␣
        ↪"blue", "green", "yellow", "purple"],
                  "height": [80, 80, 70, 100, 90, 80, 80, 70, 100, 90]}
      # Right now new_df is a dictionary, we need to convert it to a dataframe
      new_df = pd.DataFrame(new_data)

      #Now we join the two dataframes
      joined = first_10.join(new_df, how='left') # There are 4 different types of how:
        ↪ outer, inner, left, right.

      joined
```

```
[12]:    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
      0                5.1               3.5                1.4               0.2
      1                4.9               3.0                1.4               0.2
      2                4.7               3.2                1.3               0.2
      3                4.6               3.1                1.5               0.2
      4                5.0               3.6                1.4               0.2
      5                5.4               3.9                1.7               0.4
      6                4.6               3.4                1.4               0.3
      7                5.0               3.4                1.5               0.2
      8                4.4               2.9                1.4               0.2
      9                4.9               3.1                1.5               0.1
```

```
    color  height
0     red      80
1    blue      80
2   green      70
3  yellow     100
4  purple      90
5     red      80
6    blue      80
7   green      70
8  yellow     100
9  purple      90
```

Different types of how to join two data frames
This is important if your dataframes do not have the same amount of rows

left → all rows from the left DataFrame (default).

right → all rows from the right DataFrame.

inner → only rows with matching index values in both.

outer → all rows from both, fill missing with NaN.

[13]:
```
# Excercise - Which types of join (the 'how=') will work in the example above?␣
 ↪Try them out and see what happens

# Answer: All four types of join will work in this example because both␣
 ↪Dataframes have the same row indices (0 to 9).

joined_left = first_10.join(new_df, how='left')
print("left:", joined_left.shape)
display(joined_left.head())

joined_inner = first_10.join(new_df, how='inner')
print("inner:", joined_inner.shape)
display(joined_inner.head())

joined_right = first_10.join(new_df, how='right')
print("right:", joined_right.shape)
display(joined_right.head())

joined_outer = first_10.join(new_df, how='outer')
print("outer:", joined_outer.shape)
display(joined_outer.head())
```

```
left: (10, 6)
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
```

```
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

    color  height
0     red      80
1    blue      80
2   green      70
3  yellow     100
4  purple      90
```

```
inner: (10, 6)
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

    color  height
0     red      80
1    blue      80
2   green      70
3  yellow     100
4  purple      90
```

```
right: (10, 6)
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

    color  height
0     red      80
1    blue      80
2   green      70
3  yellow     100
4  purple      90
```

```
outer: (10, 6)
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
```

```
    color  height
0     red      80
1    blue      80
2   green      70
3  yellow     100
4  purple      90
```

[ ]:
```
# Excercise 2 - Add a row to one of the dataframes and see what happens when␣
 ↪you join them again
first11 = df.iloc[0:11, :]

joined_with_extra = first11.join(new_df, how='left')

joined_with_extra

right_joined_with_extra = first11.join(new_df, how='right')
right_joined_with_extra
```

[ ]:
```
    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                 5.1               3.5                1.4               0.2
1                 4.9               3.0                1.4               0.2
2                 4.7               3.2                1.3               0.2
3                 4.6               3.1                1.5               0.2
4                 5.0               3.6                1.4               0.2
5                 5.4               3.9                1.7               0.4
6                 4.6               3.4                1.4               0.3
7                 5.0               3.4                1.5               0.2
8                 4.4               2.9                1.4               0.2
9                 4.9               3.1                1.5               0.1
10                5.4               3.7                1.5               0.2

     color  height
0      red    80.0
1     blue    80.0
2    green    70.0
3   yellow   100.0
4   purple    90.0
5      red    80.0
6     blue    80.0
7    green    70.0
8   yellow   100.0
9   purple    90.0
10     NaN     NaN
```

Concatenating

You can also join two dataframes bu simply gluing them together.

```
[21]: # We just made a subset of the original dataframe called 'first_10' now we find␣
      ↪the last 10 and glue them together
      last_10 = df.iloc[-10:, :]   # selecting the last 10 rows using one of the␣
      ↪methods we learned above



      # Now we concatenate the two dataframes together
      concatenated = pd.concat( [first_10, last_10], axis=0)  # axis=0 means we␣
      ↪concatenate rows, axis=1 would concatenate columns
      concatenated
```

```
[21]:      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
      0                  5.1               3.5                1.4               0.2
      1                  4.9               3.0                1.4               0.2
      2                  4.7               3.2                1.3               0.2
      3                  4.6               3.1                1.5               0.2
      4                  5.0               3.6                1.4               0.2
      5                  5.4               3.9                1.7               0.4
      6                  4.6               3.4                1.4               0.3
      7                  5.0               3.4                1.5               0.2
      8                  4.4               2.9                1.4               0.2
      9                  4.9               3.1                1.5               0.1
      140                6.7               3.1                5.6               2.4
      141                6.9               3.1                5.1               2.3
      142                5.8               2.7                5.1               1.9
      143                6.8               3.2                5.9               2.3
      144                6.7               3.3                5.7               2.5
      145                6.7               3.0                5.2               2.3
      146                6.3               2.5                5.0               1.9
      147                6.5               3.0                5.2               2.0
      148                6.2               3.4                5.4               2.3
      149                5.9               3.0                5.1               1.8
```

Now you have played around with some of the basics manipulation in pandas! Now lets jump into some challenges

Yellow excercises

OBS: To ensure you can go back in 3 months time and read you code and understand the logics behind it it needs to be well commented.
So, while you solve the yellow excercises ensure that you add some meaningful comments about the logics and coding choices.

:))

The Yellow excercises is borrowed from last years couse and written by Ethan Weed

Music sales challenge

Write a script that:

1. Combines the tables of best-selling physical singles and best-selling digital singles on the Wikipedia page "List_of_best-selling_singles"
2. Outputs the artist and single name for the year you were born. If there is no entry for that year, take the closest year after you were born.
3. Outputs the artist and single name for the year you were 15 years old.

[22]:
```
# Starter code
#musicdata = pd.read_html("https://en.wikipedia.org/wiki/
↪List_of_best-selling_singles")
url_music = "https://en.wikipedia.org/wiki/List_of_best-selling_singles"

# Add a User-Agent header so Wikipedia doesn't block it
headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"}
response = requests.get(url_music, headers=headers)

# Pass the HTML text to pandas
musicdata = pd.read_html(response.text)



#Extracting physical and digital singles from the musicdata
physical_singles = musicdata[0]
digital_singles = musicdata[3]

physical_singles['Type'] = 'Physical'
digital_singles['Type'] = 'Digital'

# Combining the two tables
combined_singles = pd.concat([physical_singles, digital_singles])
combined_singles.head()
```

```
/var/folders/0y/tt7d7wj97hbf8tnc9hnp1_f00000gn/T/ipykernel_86483/1297234199.py:1
0: FutureWarning: Passing literal html to 'read_html' is deprecated and will be
removed in a future version. To read from a literal string, wrap it in a
'StringIO' object.
  musicdata = pd.read_html(response.text)
```

[22]:
```
                    Artist                                           Single  \
0              Bing Crosby                                "White Christmas"
1               Elton John  "Something About the Way You Look Tonight"/"Ca…
2              Bing Crosby                                    "Silent Night"
3               Tino Rossi                                "Petit Papa Noël"
4  Bill Haley & His Comets                            "Rock Around the Clock"

   Released Sales (in millions)  Source      Type
0      1942                  50     [1]  Physical
1      1997                  33     [1]  Physical
2      1935                  30     [2]  Physical
3      1946                  30     [3]  Physical
```

```
4      1954                     25   [4][5]  Physical
```

```
[ ]: # Print the artist and single from the year you were 15 years old.

     subset_15 = combined_singles[combined_singles["Released"]== 2018]
     subset_15
```

```
[ ]:                              Artist                                Single  \
     16                         Katy Perry                            "Firework"
     21   Shakira featuring Freshlyground  "Waka Waka (This Time for Africa)"

         Released Sales (in millions) Source     Type
     16      2010                    17    [59]  Digital
     21      2010                    15    [61]  Digital
```

```
[ ]: #this is just for fun and because I'm curious
     subset_artist = combined_singles[combined_singles["Artist"]== "Taylor Swift"]
     subset_artist
```

```
[ ]:          Artist         Single  Released Sales (in millions) Source     Type
     15  Taylor Swift  "Love Story"      2008                 18[a]   [58]  Digital
```

1.2 Space challenge

1. Make a single dataframe that combines the space missions from the 1950's to the 2020's
2. Write a script that returns the year with the most launches
3. Write a script that returns the most common month for launches
4. Write a script that ranks the months from most launches to fewest launches

```
[73]: # Starter code.
      url_space =  "https://en.wikipedia.org/wiki/
       ↪Timeline_of_Solar_System_exploration"

      # Add a User-Agent header so Wikipedia doesn't block it
      headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"}
      response = requests.get(url_space, headers=headers)

      # Pass the HTML text to pandas
      spacedata = pd.read_html(response.text)

      # combine all tables into data frame
      combined_space = pd.concat(spacedata, ignore_index = True)

      # Dropping column we dont need
      combined_space = combined_space.iloc[:, 0:3]
      combined_space.head()
```

```
/var/folders/0y/tt7d7wj97hbf8tnc9hnp1_f00000gn/T/ipykernel_86483/4030390085.py:9
```

```
: FutureWarning: Passing literal html to 'read_html' is deprecated and will be
removed in a future version. To read from a literal string, wrap it in a
'StringIO' object.
  spacedata = pd.read_html(response.text)
```

[73]:
```
   Mission name       Launch date  \
0     Sputnik 1    4 October 1957
1     Sputnik 2   3 November 1957
2    Explorer 1   1 February 1958
3    Vanguard 1      17 March 1958
4        Luna 1    2 January 1959

                                        Description
0                                First Earth orbiter
1  Earth orbiter, first animal in orbit, a dog na…
2  Earth orbiter; discovered Van Allen radiation …
3  Earth orbiter; oldest spacecraft still in Eart…
4  First lunar flyby (attempted lunar impact?); f…
```

[ ]:
```
## The year with the most launches

#need to clean data
combined_space['Launch Year'] = pd.to_datetime(combined_space['Launch date']).
 ↪dt.year




# Counting the number of launches per year
launch_counts = combined_space['Launch Year'].value_counts()
most_launches_year = launch_counts.idxmax()
most_launches_count = launch_counts.max()
combined_space.head()

print(f"The year with the most launches is {most_launches_year} with␣
 ↪{most_launches_count} launches.")

#there is some missing data in the launch date column, and that's why the year␣
 ↪is a float.
```

```
The year with the most launches is 1965.0 with 12 launches.
```

[81]:
```
# The month with the most launches
months = {
    1: "January", 2: "February", 3: "March", 4: "April",
    5: "May", 6: "June", 7: "July", 8: "August",
    9: "September", 10: "October", 11: "November", 12: "December"
}
```

```
combined_space['Launch Month'] = pd.to_datetime(combined_space['Launch date']).
  ↪dt.month
launch_month_counts = combined_space['Launch Month'].value_counts()
most_launches_month = launch_month_counts.idxmax()
most_launches_month_count = launch_month_counts.max()
print(f"The month with the most launches is {months[most_launches_month]} with␣
  ↪{most_launches_month_count} launches.")
```

```
The month with the most launches is November with 30 launches.
```

[83]:
```
# Ranking of months with the most to the fewest launches
# Sorting the counts in descending order
sorted_month_counts=launch_month_counts.sort_values(ascending=False)
print("Ranking of months with the most to the fewest launches:")
for month, count in sorted_month_counts.items():
    print(f"{months[month]}:{count} launches")
```

```
Ranking of months with the most to the fewest launches:
November:30 launches
August:27 launches
September:25 launches
October:24 launches
January:21 launches
July:21 launches
December:19 launches
February:18 launches
May:18 launches
March:15 launches
June:14 launches
April:13 launches
```

1.3 Supervillain challenge

1. Write a script that combines the tables showing supervillain debuts from the 30's through the 2010's
2. Write a script that ranks each decade in terms of how many supervillains debuted in that decade
3. Write a script that ranks the different comics companies in terms of how many supervillains they have, and display the results in a nice table (pandas dataframe)

[94]:
```
#supervillandata = pd.read_html("https://en.wikipedia.org/wiki/
  ↪List_of_comic_book_supervillain_debuts")

url_villan = "https://en.wikipedia.org/wiki/
  ↪List_of_comic_book_supervillain_debuts"

# Add a User-Agent header so Wikipedia doesn't block it
```

```
headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"}
response = requests.get(url_villan, headers=headers)

# Pass the HTML text to pandas
supervillandata = pd.read_html(response.text)

# combine all tables into data frame
df_supervillan = pd.concat(supervillandata, ignore_index = True)
df_supervillan[:10]
```

```
/var/folders/0y/tt7d7wj97hbf8tnc9hnp1_f00000gn/T/ipykernel_86483/4250549170.py:1
0: FutureWarning: Passing literal html to 'read_html' is deprecated and will be
removed in a future version. To read from a literal string, wrap it in a
'StringIO' object.
  supervillandata = pd.read_html(response.text)
```

```
[94]:     0                                                  1 Character / Team  \
      0 NaN  This article has multiple issues. Please help …              NaN
      1 NaN  This article includes a list of general refere…              NaN
      2 NaN  This article needs additional citations for ve…              NaN
      3 NaN  This article includes a list of general refere…              NaN
      4 NaN  This article needs additional citations for ve…              NaN
      5 NaN                                                NaN   Ultra-Humanite
      6 NaN                                                NaN        Dr. Death
      7 NaN                                                NaN         The Monk
      8 NaN                                                NaN         The Claw
      9 NaN                                                NaN         Hath-Set

         Year Debuted                    Company                    Creator/s  \
      0             NaN                        NaN                          NaN
      1             NaN                        NaN                          NaN
      2             NaN                        NaN                          NaN
      3             NaN                        NaN                          NaN
      4             NaN                        NaN                          NaN
      5     1939 (June)                         DC    Jerry Siegel, Joe Shuster
      6     1939 (July)                         DC         Bob Kane, Bill Finger
      7  1939 (September)                       DC         Bob Kane, Bill Finger
      8   1939 (December)  Lev Gleason Publications                    Jack Cole
      9    1940 (January)                        DC  Gardner Fox, Dennis Neville

                   First Appearance
      0                          NaN
      1                          NaN
      2                          NaN
      3                          NaN
      4                          NaN
      5     Action Comics (vol. 1) #13
```

```
6  Detective Comics (vol. 1) #29
7  Detective Comics (vol. 1) #31
8        Silver Streak Comics #1
9                Flash Comics #1
```

```
[ ]: # 1. Write a script that combines the tables showing supervillain debuts from␣
      ↪the 30's through the 2010's

     #remove rows if year debuted is Na-s
     clean = df_supervillan.dropna(subset=['Year Debuted']).copy()


     # 2. Which decade had the most supervillain debuts?
     #make year deputed clean, splitting the year from the rest of the string
     clean['Year'] = clean['Year Debuted'].str.extract(r'(\d{4})')

     clean['Year'] = pd.to_numeric(clean['Year'])


     #find min and max value of year
     min_year = decade_counts['Year'].min()
     max_year = decade_counts['Year'].max()
     print(f"The minimum year is {min_year} and the maximum year is {max_year}")


     #group by decade
     clean['Decade'] = (clean['Year'] // 10) * 10

     clean.head()
```

```
The minimum year is 1939 and the maximum year is 2019
```

```
[ ]:     0    1 Character / Team      Year Debuted                  Company  \
     5 NaN  NaN   Ultra-Humanite       1939 (June)                       DC
     6 NaN  NaN        Dr. Death       1939 (July)                       DC
     7 NaN  NaN         The Monk  1939 (September)                       DC
     8 NaN  NaN         The Claw   1939 (December)  Lev Gleason Publications
     9 NaN  NaN         Hath-Set    1940 (January)                       DC

                        Creator/s             First Appearance  Year  Decade
     5   Jerry Siegel, Joe Shuster     Action Comics (vol. 1) #13  1939    1930
     6         Bob Kane, Bill Finger  Detective Comics (vol. 1) #29  1939    1930
     7         Bob Kane, Bill Finger  Detective Comics (vol. 1) #31  1939    1930
     8                     Jack Cole        Silver Streak Comics #1  1939    1930
     9  Gardner Fox, Dennis Neville                Flash Comics #1  1940    1940
```

[ ]:
```python
# 2. Write a script that ranks each decade in terms of how many supervillains␣
 ↪debuted in that decade

def decade_with_most_debuts(df):
    decade_counts = df['Decade'].value_counts().sort_index()
    most_debuts_decade = decade_counts.idxmax()
    most_debuts_count = decade_counts.max()
    return most_debuts_decade, most_debuts_count, decade_counts

most_debuts_decade, most_debuts_count, decade_counts =␣
 ↪decade_with_most_debuts(clean)
print(f"The decade with the most supervillain debuts is the␣
 ↪{most_debuts_decade}s with {most_debuts_count} debuts.")
#print in order
print("Ranking of decades by supervillain debuts:")
for decade, count in decade_counts.sort_values(ascending=False).items():
    print(f"{decade}s: {count} ")
```

```
The decade with the most supervillain debuts is the 1960s with 228 debuts.
Ranking of decades by supervillain debuts:
1960s: 228
1970s: 97
1980s: 91
1990s: 84
2000s: 50
1940s: 47
1950s: 26
2010s: 14
1930s: 4
```

[119]:
```python
# 3. Write a script that ranks the different comics companies in terms of how␣
 ↪many supervillains they have, and display the results

company_counts = clean['Company'].value_counts()
print("Ranking of comic companies by number of supervillains:")
for company, count in company_counts.items():
    print(f"{company}: {count} ")
```

```
Ranking of comic companies by number of supervillains:
DC: 337
Marvel: 270
Fawcett Comics/DC: 6
Image: 5
Dark Horse: 5
Marvel/Timely: 4
Disney/Hyperion: 4
Eternity: 3
Lev Gleason Publications: 1
```

```
Comico: 1
Mirage: 1
Image Comics: 1
```