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]:
```
# 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

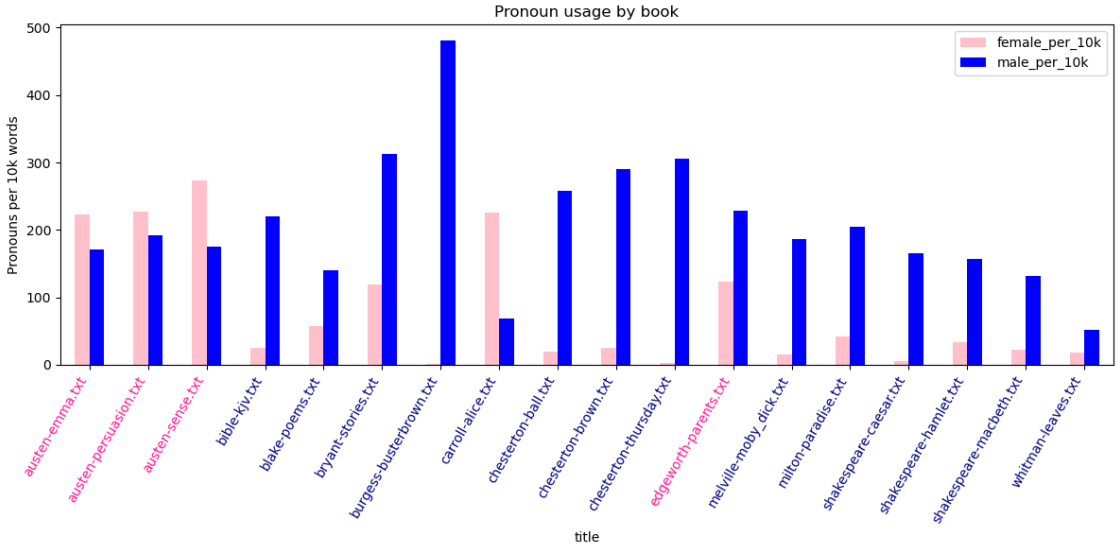
```
15   shakespeare-hamlet.txt    male                125                588
16  shakespeare-macbeth.txt    male                 53                306
17       whitman-leaves.txt    male                278                809
```

[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']
```

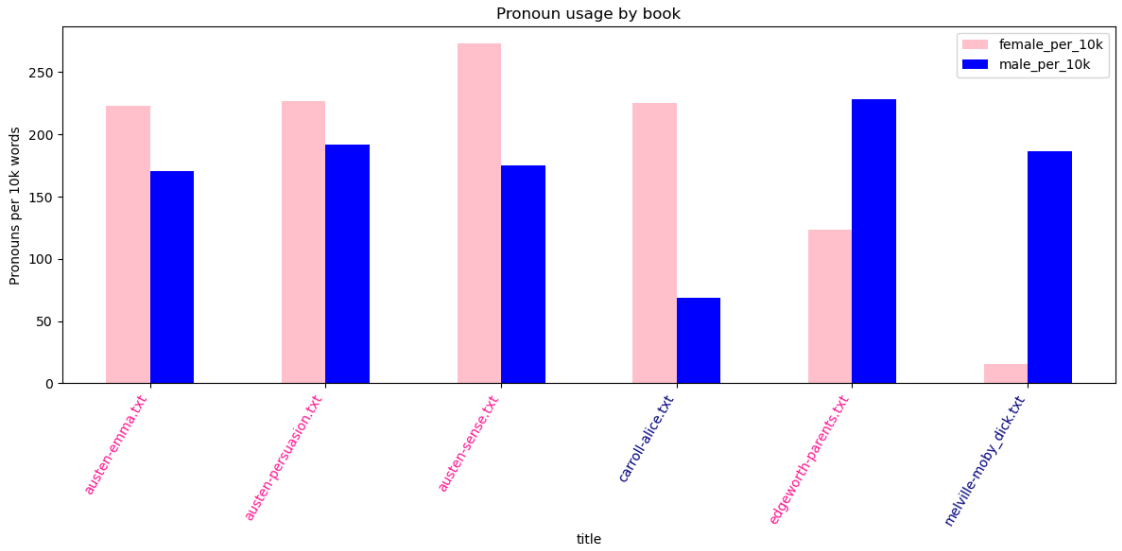
[67]:
```
#visualise this data - Also: normalise the distribution, because the bible is␣
 ↪much longer - higher overall pronoun count
import matplotlib.pyplot as plt
# Add a column with total number of tokens (words) in each book
df["tokens"] = df["title"].apply(lambda t: len(nltk.corpus.gutenberg.words(t)))

# 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.