Cultural Data Science Portfolio Exam

Markus Lundsfryd Jensen (MJ), au755063@uni.au.dk

School of Communication and Culture, Aarhus University

Jens Chr. Skous Vej 2, 8000 Aarhus C, Denmark

Supervisor: Anna Zamm

Table of Contents

Please Note that Assignments 3-5 have been made in corroboration with Rune Egeskov Trust, au692561@uni.au.dk. Individual contributions have been denoted with the initials (RT) for Rune and (MJ) for Markus or their first names.

Assignment 01

- *Basics of programming and data organization*

Link to Github repository: https://github.com/Lundsfryd/CulturalDataScience.git

Week 2 - Variables

Author: Markus Lundsfryd Jensen
Date: 17.12.2025

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

Variables and storing data in them

Fill in the blanks:

Which variable does these data types belong to?
Whole numbers = integer Decimal numbers = float Collections of characters = string True or False values = Boolean

In [1]:
```
# 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 = 1.4
print(type(x))

name = "Markus"
print(type(name))

this_is_fun = False
print(type(this_is_fun))

height = 134

print(type(height))
```

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

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.

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

# list

# dictionary

# tuple

# set
```

Acessing elements

Access a value from each container variable you just created

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

# dictionary - Access values by their key.

# tuple - Access elements by index, just like lists.

# set - Cannot access items by index because sets are unordered. but you car
```

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:

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:

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:

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:

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.

In [ ]:
```
# install the natural language toolkit package (nltk), which has a copy of t

#%pip install nltk notebook nbconvert
```

In [ ]:
```
# import the nltk package so that it is accessible to Python, and download a
import nltk
#nltk.download('gutenberg')
```

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

In [4]:
```
# 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 tex
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?

By splitting by whitespace each word is treated as a individual entity instead of a bunch of characters (one string)

In [5]:
```
# make a variable called "concordance", and fill it with every occurrence of
lexicon = []
for i, val in enumerate(mobydick):
    if val == "whale":
            lexicon.append(str(' '.join(mobydick[i-5:i+5])))
```

In [8]:
```
# take a look at what the algorithm has found
#lexicon
```

In [6]:
```
# let's see how many instances of the word "whale" were found
len(lexicon)
```

Out[6]: 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

In [9]:
```
lexicon_twowords = [] #initialize empty list
for i, val in enumerate(mobydick): #for i (index) of value in numbered value
    if val == "whale": #if the value of the string is equal to "whale"
        if mobydick[i-1] == "the": #and the preceding value is "the"
            lexicon_twowords.append(str(' '.join(mobydick[i-5:i+5]))) #to th
                                                                      #(and
```

In [10]:
```
# take a look at what the algorithm has found

print(f'number of occorences: {len(lexicon_twowords)}')
#print(lexicon_twowords)
```

```
number of occorences: 175
```

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.

In [11]:
```
nltk.corpus.gutenberg.fileids()
```

Out[11]:
```
['austen-emma.txt',
 'austen-persuasion.txt',
 'austen-sense.txt',
 'bible-kjv.txt',
 'blake-poems.txt',
 'bryant-stories.txt',
 'burgess-busterbrown.txt',
 'carroll-alice.txt',
 'chesterton-ball.txt',
 'chesterton-brown.txt',
 'chesterton-thursday.txt',
 'edgeworth-parents.txt',
 'melville-moby_dick.txt',
 'milton-paradise.txt',
 'shakespeare-caesar.txt',
 'shakespeare-hamlet.txt',
 'shakespeare-macbeth.txt',
 'whitman-leaves.txt']
```

In [12]:
```
# Add your code to build your own lexicon here, using the same method as abo

bible = nltk.corpus.gutenberg.raw("bible-kjv.txt") #save bible text in varia

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

bible = bible.replace('\n', ' ') #replace line shifts with whitespace
bible = bible.replace('\r', ' ') #--//--

bible = bible.split()
```

In [13]:
```
death_lexicon = ["death","execute", "die"]
death_words_bible = []

for i, word in enumerate(bible):
    if word in death_lexicon:
        death_words_bible.append(word)

    else:
        pass
```

In [16]:
```
print(len(death_words_bible))
print(death_words_bible[:5])
```

```
269
['death', 'death', 'die', 'death', 'die']
```

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.

In [18]:
```
# One way to do it: Print all the titles and numbers of words
# starter code:

books = nltk.corpus.gutenberg.fileids()

number_of_words = 0

for i, title in enumerate(books):
    book = nltk.corpus.gutenberg.raw(title)
    book = book.lower()
    book = book.replace('\n', ' ') #replace line shifts with whitespace
    book = book.replace('\r', ' ') #--//--
    book = book.split()

    if len(book)>number_of_words:
        print(f'this is the book with most words {title}. Number of words: {
        number_of_words = len(book)
```

```
this is the book with most words austen-emma.txt. Number of words: 158167
this is the book with most words bible-kjv.txt. Number of words: 821133
```

In [19]:
```
def word_counter(title):
    book = nltk.corpus.gutenberg.raw(title)
    book = book.lower()
    book = book.replace('\n', ' ') #replace line shifts with whitespace
    book = book.replace('\r', ' ') #--//--
    book = book.split()

    return len(book)
```

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

In [ ]:
```
# Another way to do it: Make a list of titles and a list of wordcounts, put
#%pip install pandas
import pandas as pd


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

titles = []
numwords = []
for title in books:
    nr_words = word_counter(title)
```

```
    numwords.append(nr_words)
    titles.append(title)
```

In [21]:
```
data = {"titles":titles, "number_of_words": numwords}

df = pd.DataFrame(data = data)

df.sort_values("number_of_words", ascending = False)
```

Out[21]:

	titles	number_of_words
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

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?

In [22]:
```
# starter code:

#book = nltk.corpus.gutenberg.raw('yourbook.txt')
#words = book.lower()


freq = nltk.FreqDist(bible)

top_ten = freq.most_common(10)
print(top_ten)
```

```
[('the', 64014), ('and', 51313), ('of', 34634), ('to', 13567), ('that', 1278
4), ('in', 12503), ('he', 10261), ('shall', 9838), ('unto', 8987), ('for', 8
810)]
```

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.

In [23]:
```
# list of stopwords

stopwords = ["", "i", "me", "my", "myself", "we", "our", "ours", "ourselves"

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

freq = nltk.FreqDist(words)

top_ten = freq.most_common(10)
print(top_ten)
```

```
[('shall', 9838), ('unto', 8987), ('thou', 5202), ('lord', 4739), ('thy', 46
00), ('ye', 3848), ('upon', 2734), ('god', 2303), ('said', 2285), ('hath', 2
242)]
```

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 :)

In [24]:
```
def split_into_words(title):
    book = nltk.corpus.gutenberg.raw(title)
    book = book.lower()
    book = book.replace('\n', ' ') #replace line shifts with whitespace
    book = book.replace('\r', ' ') #--//--
    book = book.split()
    return book


def word_counter(title):
    book = split_into_words(title)
    return len(book)

def pronoun_extracter(title, pronouns):

    book = split_into_words(title)
    pronoun_words = [word for word in book if word in pronouns]

    return pronoun_words
```

In [25]:
```
# Example of structure:

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

# Male pronouns
male_pronouns = {
    "he", "him", "his", "himself"
}

# Female pronouns
female_pronouns = {
    "she", "her", "hers", "herself"
}

# Make a dictionary of all titles in the corpus and the authors gender (1 di
```

```
author_genders = {
    "austen-emma.txt": "female",        # Jane Austen
    "austen-persuasion.txt": "female",  # Jane Austen
    "austen-sense.txt": "female",       # Jane Austen
    "bible-kjv.txt": "male",            # Multiple (traditionally male autho
    "blake-poems.txt": "male",          # William Blake
    "bryant-stories.txt": "male",       # William Cullen Bryant
    "burgess-busterbrown.txt": "male",  # Thornton W. Burgess
    "carroll-alice.txt": "male",        # Lewis Carroll
    "chesterton-ball.txt": "male",      # G. K. Chesterton
    "chesterton-brown.txt": "male",     # G. K. Chesterton
    "chesterton-thursday.txt": "male",  # G. K. Chesterton
    "edgeworth-parents.txt": "female",  # Maria Edgeworth
    "melville-moby_dick.txt": "male",   # Herman Melville
    "milton-paradise.txt": "male",      # John Milton
    "shakespeare-caesar.txt": "male",   # William Shakespeare
    "shakespeare-hamlet.txt": "male",   # William Shakespeare
    "shakespeare-macbeth.txt": "male",  # William Shakespeare
    "whitman-leaves.txt": "male",       # Walt Whitman
}

# Make empty lists to store title, genders, number of female pronouns and nu
book_data = []
# Make a loop that iterates over each book, splits the text to words, counts

books = nltk.corpus.gutenberg.fileids()

print("initializing forloop")
for book in books:

    male_pronouns = pronoun_extracter(book, male_pronouns)
    female_pronouns = pronoun_extracter(book, female_pronouns)

    data = {"title": book, "author_gender": author_genders[book], "male_pror
    book_data.append(data)
    print(f'appended data for: {book}')
```

```
initializing forloop
appended data for: austen-emma.txt
appended data for: austen-persuasion.txt
appended data for: austen-sense.txt
appended data for: bible-kjv.txt
appended data for: blake-poems.txt
appended data for: bryant-stories.txt
appended data for: burgess-busterbrown.txt
appended data for: carroll-alice.txt
appended data for: chesterton-ball.txt
appended data for: chesterton-brown.txt
appended data for: chesterton-thursday.txt
appended data for: edgeworth-parents.txt
appended data for: melville-moby_dick.txt
appended data for: milton-paradise.txt
appended data for: shakespeare-caesar.txt
appended data for: shakespeare-hamlet.txt
appended data for: shakespeare-macbeth.txt
appended data for: whitman-leaves.txt
```

In [26]:
```
# Make dataframe, fill it with the lists above, and print it

df_books = pd.DataFrame(book_data)
df_books
```

Out[26]:

	title	author_gender	male_pronouns	female_pronouns
0	austen-emma.txt	female	3284	4292
1	austen-persuasion.txt	female	1884	2223
2	austen-sense.txt	female	2477	3864
3	bible-kjv.txt	male	22210	2530
4	blake-poems.txt	male	117	48
5	bryant-stories.txt	male	1741	656
6	burgess-busterbrown.txt	male	911	3
7	carroll-alice.txt	male	233	726
8	chesterton-ball.txt	male	2509	186
9	chesterton-brown.txt	male	2493	217
10	chesterton-thursday.txt	male	2115	15
11	edgeworth-parents.txt	female	4814	2553
12	melville-moby_dick.txt	male	4856	391
13	milton-paradise.txt	male	1978	406
14	shakespeare-caesar.txt	male	426	14
15	shakespeare-hamlet.txt	male	588	125
16	shakespeare-macbeth.txt	male	306	53
17	whitman-leaves.txt	male	785	274

Discuss:

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

Week 3 - Loops, Conditions and Functions

Author: Markus Lundsfryd Jensen Date: 17.12.2025

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!

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

Green excercises

Conditional Statements

The *if* statement

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

In [2]:
```
# 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
```

In [ ]:
```
# Exceise: Modify the code to check if x is less than 5 and print a differer
```

The *else* statement

In [ ]:
```
# Basic syntax

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

In [3]:
```
# 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
```

In [ ]:
```
# Excercise - Change the code above so that it prints "x is greater than 5."
# How many solutions can you find? and is some of them better than others?
```

The *elif* statement (for multiple conditions)

In [ ]:
```
# Basic syntax

# if condition1:
    # code if condition1 is True
# elif condition2:
    # code if condition2 is True
# else:
```

```
    # code if neither condition is True
```

In [4]:
```
# 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
```

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

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.

In [5]:
```
# 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
```

In [ ]:
```
# Exercise - Modify the code to print the length of each fruit instead of th
```

In [6]:
```
# 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
```

In [ ]:
```
# Excercise - Modify the code to count up from 1 to 5 instead of counting do
```

Functions

In [9]:
```
# Heres a calculator
def add_two_numbers(a, b):
    result = a + b
    print(f"{a} + {b} = {result}")
    return result
```

In [10]:
```
# Try it out with different numbers
add_two_numbers(5,5)
```

```
5 + 5 = 10
```

Out[10]: 10

In [ ]:
```
# Make a subtraction function
```

Yellow excercises

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: ")**

In [11]:
```
def password_checker():
    while True:
        psw = input('Enter your password:')
        psw_list = list(psw)
        psw_length = len(psw_list)

        if psw_length <= 32 and psw_length >=8:
            print('all good')
```

```
            break
        else:
            print(f'pasword must be between 8 and 32 charecters.\nCurrent pa

    return
```

In [12]:
```
password_checker()
```

```
pasword must be between 8 and 32 charecters.
Current pasword is 5 charecters.
 try again:

all good
```

Now lets add some extra requirements:

- The password needs to have at least 3 digits
- And there needs to be an uppercase letter

In [13]:
```
def password_checker():
    while True:
        psw = input('Enter your password:')
        psw_list = list(psw)
        psw_length = len(psw_list)

        if psw_length <= 32 and psw_length >=8:

            digits = digit_checker(psw_list)

            if len(digits)>= 3:
                print('all good')
                break
            else:
                print(f'pasword must have at least three digits.\nCurrent pa


        else:
            print(f'pasword must be between 8 and 32 charecters.\nCurrent pa

    return


def digit_checker(psw):
    digits = []
    for chr in psw:
        try:
            integer = int(chr)
            digits.append(integer)
        except:
            pass

    return digits
```

In [14]:
```
password_checker()
```

```
pasword must be between 8 and 32 charecters.
Current pasword is 5 charecters.
 try again:

pasword must have at least three digits.
Current pasword has 0 digits.
 try again:

all good
```

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.

In [15]:
```
# Last modifications on the code - final password_checker

def password_checker():
    while True:
        psw = input('Enter your password:')
        psw_list = list(psw)
        psw_length = len(psw_list)

        if psw_length <= 32 and psw_length >=8:

            digits = digit_checker(psw_list)

            if len(digits)>= 3:

                sp_chr = special_character_checker(psw)

                if len(sp_chr) >=3:
                    print('all good')
                    break

                else:
                    print(f'pasword must have at least three special charact



            else:
                print(f'pasword must have at least three digits.\nCurrent pa


        else:
            print(f'pasword must be between 8 and 32 charecters.\nCurrent pa

    return


def digit_checker(psw):
    digits = []
    for chr in psw:
```

```
        try:
            integer = int(chr)
            digits.append(integer)
        except:
            pass

    return digits


def special_character_checker(psw):

    special_characters = "!@#$%^&*()-+?_=,<>/"

    sp_chrs = []

    for chr in psw:
        if chr in special_characters:
            sp_chrs.append(chr)

    return sp_chrs
```

In [16]:
```
password_checker()
```

```
pasword must be between 8 and 32 charecters.
Current pasword is 5 charecters.
 try again:

pasword must have at least three digits.
Current pasword has 0 digits.
 try again:

pasword must have at least three special characters.
Current pasword has 0 special characters.
 try again:

pasword must have at least three digits.
Current pasword has 0 digits.
 try again:

pasword must have at least three special characters.
Current pasword has 1 special characters.
 try again:

all good
```

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

In [17]:
```
# celsius to fahrenheit converter

def C_to_F(C):

    print(f'converting {C} degrees C to F')

    F = (C*9/5) + 32

    return F
```

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

In [18]:
```
# Fahrenheit to Celsius converter

def F_to_C(F):

    print(f'converting {F} degrees F to C')

    C = (F-32)/(9/5)

    return C
```

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.

In [19]:
```
# Temperature converter here


def temperature_converter(temp, unit = 'F'):

    if unit == "F":
        converted_temp = F_to_C(temp)

    elif unit == "C":
        converted_temp = C_to_F(temp)

    else:
        raise ValueError("Only units allowed are 'C' or 'F'")

    return converted_temp
```

In [20]:
```
temperature_converter(32,"F")
```

```
converting 32 degrees F to C
```

Out[20]:
```
0.0
```

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

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.

In [21]:
```python
# Make a function that calculate the average of test scores.
def calculate_average(test_scores):

    acc = 0

    for num in test_scores:
        acc += num

    average = acc/len(test_scores)

    return average
```

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

In [22]:
```
# Make a function that converst the test scores to letter grades

def percentage_to_letter_grade(test_score):
    grades = {9: "A", 8: "B", 7: "C", 6: "D"}

    assigned_grade = grades.get(test_score // 10, "F")
    return assigned_grade

# Test with different percentages

percentage_to_letter_grade(89)
```

Out[22]: 'B'

Last task, We need to calculate the weighted grade

In [23]:
```
def calculate_weighted_grade(test_score, homework_score):

    weighted_grade = (test_score*0.7) + (homework_score *0.3)
    return weighted_grade
```

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.

In [24]:
```
def calculate_student_grade(test_scores, homework_scores):

    aver_test = calculate_average(test_scores)
    aver_homework = calculate_average(homework_scores)

    weighted_score = calculate_weighted_grade(aver_test, aver_homework)

    letter_grade = percentage_to_letter_grade(weighted_score)

    return letter_grade, weighted_score
```

In [26]:
```
test_stuff = [85, 93, 78, 96, 88]
home_stuff = [80,100,90]
calculate_student_grade(test_stuff, home_stuff)
```

Out[26]: ('B', 88.6)

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

In [27]:
```python
# Code here :)

class StudentGrader(object):

    def __init__(self, test_scores_list, homework_scores_list):

        self.test_scores_list = test_scores_list
        self.homework_scores_list = homework_scores_list

        return


    def calculate_average(selt, test_scores):
        '''
        # Make a function that calculate the average of test scores.
        '''

        acc = 0

        for num in test_scores:
            acc += num

        average = acc/len(test_scores)

        return average


    def calculate_weighted_grade(self, test_score, homework_score):

        weighted_grade = (test_score*0.7) + (homework_score *0.3)
        return weighted_grade



    def percentage_to_letter_grade(self, test_score):
        '''
        # Make a function that converst the test scores to letter grades
        '''
        grades = {9: "A", 8: "B", 7: "C", 6: "D"}

        assigned_grade = grades.get(test_score // 10, "F")
        return assigned_grade



    def calculate_student_grade(self):

        '''
        main function
```

```
        '''

        aver_test = self.calculate_average(self.test_scores_list)
        aver_homework = self.calculate_average(self.homework_scores_list)

        weighted_score = self.calculate_weighted_grade(aver_test, aver_homew

        letter_grade = self.percentage_to_letter_grade(weighted_score)

        return letter_grade, weighted_score
```

In [28]:
```
test_stuff = [85, 93, 78, 96, 88]
home_stuff = [80,100,90]
SG = StudentGrader(test_stuff, home_stuff)

output = SG.calculate_student_grade()

print(output)
```

```
('B', 88.6)
```

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.

Week 4 - Data organising with Pandas

Author: Markus Lundsfryd Jensen Date: 17.12.2025

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!

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

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

Green excercises

Data organisation using a dataset about flowers

In [2]:
```
flower = datasets.load_iris()

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

df.head()
```

Out[2]:

	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

In [3]:
```
# There are some commands in the library pandas that can give you a quick ov

df.head()      # first 5 rows, if you put a number into the paranthesis you
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
```

Out[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Selecting columns and rows

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

In [4]:
```
# 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 ir
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
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

In [5]:
```
# Let's try to select some data using conditionals

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

Out[5]:

	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

In [6]:
```
# Here we select all rows where the sepal length is larger than or equal to
length_and_width = df[(df["sepal length (cm)"] >= 5) & (df["sepal width (cm)
length_and_width.head()
```

Out[6]:

	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

In [7]:
```
import numpy as np
# Excercise - Find the longest petal length and the median petal length
longest_pl = df['sepal length (cm)'].max()
median_pl = df['sepal length (cm)'].median()
# and subset the flowers that are between the median and one centimeter shor

# Get indices that meets requirements

#one option
subset_1 = df[(df["sepal length (cm)"] >= median_pl) & (df["sepal width (cm)
```

Sorting data

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

In [8]:
```
# we could sort the data by a specific column in both ascending and descendi
df_sorted = df.sort_values(by="sepal length (cm)", ascending=False) # change
df_sorted.head()
```

Out[8]:

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

In [9]:
```
# Excercise - sort the data by petal width in ascending order and select the
df_sorted = df.sort_values(by="sepal width (cm)", ascending=True) # change c

df_sorted.head(10)
```

Out[9]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
60	5.0	2.0	3.5	1.0
62	6.0	2.2	4.0	1.0
119	6.0	2.2	5.0	1.5
68	6.2	2.2	4.5	1.5
87	6.3	2.3	4.4	1.3
93	5.0	2.3	3.3	1.0
53	5.5	2.3	4.0	1.3
41	4.5	2.3	1.3	0.3
57	4.9	2.4	3.3	1.0
80	5.5	2.4	3.8	1.1

Flipping

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

In [10]:
```
print(df.head(5))

reversed_df = df.iloc[::-1]   # Flipping the dataframe horisontally (reverse

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 (c
m)
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.

In [11]:
```
# Joining a bit of the iris data with a new dataframe (we will make up some
first_10 = df.iloc[0:10, :]  # selecting the first 10 rows of the iris data
new_data = {"color": ["red", "blue", "green", "yellow", "purple", "red", "bl
            "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

joined
```

Out[11]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	color	height
0	5.1	3.5	1.4	0.2	red	80
1	4.9	3.0	1.4	0.2	blue	80
2	4.7	3.2	1.3	0.2	green	70
3	4.6	3.1	1.5	0.2	yellow	100
4	5.0	3.6	1.4	0.2	purple	90
5	5.4	3.9	1.7	0.4	red	80
6	4.6	3.4	1.4	0.3	blue	80
7	5.0	3.4	1.5	0.2	green	70
8	4.4	2.9	1.4	0.2	yellow	100
9	4.9	3.1	1.5	0.1	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.

In [12]:
```
# Excercise - Which types of join (the 'how=') will work in the example abov

'''All will work as they ahve the same number of rows'''
```

Out[12]:
```
'All will work as they ahve the same number of rows'
```

In [13]:
```
# Excercise 2 - Add a row to one of the dataframes and see what happens wher

first_11 = df.iloc[0:11, :]  # selecting the first 10 rows of the iris data
new_data_1 = {"color": ["red", "blue", "green", "yellow", "purple", "red", "
            "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_1 = pd.DataFrame(new_data_1)

#Now we join the two dataframes
joined_1 = first_11.join(new_df_1, how='outer') # There are 4 different type

joined_1
```

Out[13]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	color	height
0	5.1	3.5	1.4	0.2	red	80.0
1	4.9	3.0	1.4	0.2	blue	80.0
2	4.7	3.2	1.3	0.2	green	70.0
3	4.6	3.1	1.5	0.2	yellow	100.0
4	5.0	3.6	1.4	0.2	purple	90.0
5	5.4	3.9	1.7	0.4	red	80.0
6	4.6	3.4	1.4	0.3	blue	80.0
7	5.0	3.4	1.5	0.2	green	70.0
8	4.4	2.9	1.4	0.2	yellow	100.0
9	4.9	3.1	1.5	0.1	purple	90.0
10	5.4	3.7	1.5	0.2	NaN	NaN

left join)

will take all 11 rows of the iris dataset and join it with new dataser, but since there is no data availbale for the last row, NA's will be value for color and height.

right join)

since there is only 10 rows in this dataset, the resulting joined dataframe will have 10 rows

inner join)

same as right join

outer join)

same as left join

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

In [14]:
```
# We just made a subset of the original dataframe called 'first_10' now we t
last_10 = df.iloc[-10:, :]   # selecting the last 10 rows using one of the m


# Now we concatenate the two dataframes together
concatenated = pd.concat( [first_10, last_10], axis=0)  # axis=0 means we co
concatenated
```

Out[14]:

	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.

In [15]:
```
def combine_singles():
    # Starter code
    #musicdata = pd.read_html("https://en.wikipedia.org/wiki/List_of_best-se
    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()

    return combined_singles
```

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.

In [16]:
```
def best_seller_singles(year_of_birth):
    '''
    function that combines singles,
    asks for year of birth,

    Output:
    artist and single name for the year you were born (or next available ent

    and
```

```
    artist and single name for the year you were 15 years old.

    '''
    combined_singles = combine_singles()

    #year_of_birth = input('Enter your year of birth:')

    single, artist, year = get_single(year_of_birth, combined_singles)

    print(f'this is the most selling single: {single} \nfrom artist: {artist

    return




def get_single(year, data):

    while True:
        row = data[data['Released'] == year]

        if  not row.empty:
            break

        print(f'no entry from year: {year}')
        year +=1
        print(f'trying next year: {year}\n')


    artist = row['Artist'].values
    single = row['Single'].values
    return single, artist, str(year)
```

In [17]:
```
best_seller_singles(2000)
```

```
no entry from year: 2000
trying next year: 2001

no entry from year: 2001
trying next year: 2002

no entry from year: 2002
trying next year: 2003

no entry from year: 2003
trying next year: 2004

no entry from year: 2004
trying next year: 2005

no entry from year: 2005
trying next year: 2006

no entry from year: 2006
trying next year: 2007

no entry from year: 2007
trying next year: 2008

this is the most selling single: ['"Love Story"']
from artist: ['Taylor Swift']
from year: 2008
```

```
/tmp/ipykernel_2271/182025985.py:11: FutureWarning: Passing literal html to
'read_html' is deprecated and will be removed in a future version. To read f
rom a literal string, wrap it in a 'StringIO' object.
  musicdata = pd.read_html(response.text)
```

In [18]:
```
best_seller_singles(2015)
```

```
this is the most selling single: ['"See You Again"' '"Uptown Funk"' '"Thinki
ng Out Loud"']
from artist: ['Wiz Khalifa featuring Charlie Puth' 'Mark Ronson featuring Br
uno Mars'
 'Ed Sheeran']
from year: 2015
```

```
/tmp/ipykernel_2271/182025985.py:11: FutureWarning: Passing literal html to
'read_html' is deprecated and will be removed in a future version. To read f
rom a literal string, wrap it in a 'StringIO' object.
  musicdata = pd.read_html(response.text)
```

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

In [19]:
```
# Starter code.
url_space =  "https://en.wikipedia.org/wiki/Timeline_of_Solar_System_explora

# 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()
```

```
/tmp/ipykernel_2271/4091429415.py:9: FutureWarning: Passing literal html to
'read_html' is deprecated and will be removed in a future version. To read f
rom a literal string, wrap it in a 'StringIO' object.
  spacedata = pd.read_html(response.text)
```

Out[19]:

	Mission name	Launch date	Description
0	Sputnik 1	4 October 1957	First Earth orbiter
1	Sputnik 2	3 November 1957	Earth orbiter, first animal in orbit, a dog na...
2	Explorer 1	1 February 1958	Earth orbiter; discovered Van Allen radiation ...
3	Vanguard 1	17 March 1958	Earth orbiter; oldest spacecraft still in Eart...
4	Luna 1	2 January 1959	First lunar flyby (attempted lunar impact?); f...

In [20]:
```
## The year with the most launches

def launch_year(data, str_index = 2, column = 'Launch date'):
    '''
    extract year and append to data
    '''

    data['Year'] = data[column].str.split().str[str_index]


    return


def launch_month(data, column = 'Launch date'):

    data['Month'] = data[column].str.split().str[1]
    return
```

```
launch_year(combined_space)
launch_month(combined_space)
```

In [21]:
```
#the year with most counts

ymc = combined_space['Year'].value_counts()[0:1]

ymc
```

Out[21]:
```
Year
1967    12
Name: count, dtype: int64
```

In [22]:
```
# The month with the most launches
mmc = combined_space['Month'].value_counts()[0:1]

mmc
```

Out[22]:
```
Month
November    31
Name: count, dtype: int64
```

In [23]:
```
# Ranking of months with the most to the fewest launches
combined_space['Month'].value_counts()
```

Out[23]:
```
Month
November     31
August       27
September    25
October      24
January      21
July         21
December     19
February     18
May          18
March        15
June         14
April        13
Name: count, dtype: int64
```

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)

In [24]:
```
#supervillandata = pd.read_html("https://en.wikipedia.org/wiki/List_of_comic

url_villan = "https://en.wikipedia.org/wiki/List_of_comic_book_supervillain_

# 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
```

```
/tmp/ipykernel_2271/1755043887.py:10: FutureWarning: Passing literal html to
'read_html' is deprecated and will be removed in a future version. To read f
rom a literal string, wrap it in a 'StringIO' object.
  supervillandata = pd.read_html(response.text)
```

Out[24]:

	0	1	Character / Team	Year Debuted	Company	Creator/s	First Appearance
0	NaN	This article has multiple issues. Please help ...	NaN	NaN	NaN	NaN	NaN
1	NaN	This article includes a list of general refere...	NaN	NaN	NaN	NaN	NaN
2	NaN	This article needs additional citations for ve...	NaN	NaN	NaN	NaN	NaN
3	NaN	This article includes a list of general refere...	NaN	NaN	NaN	NaN	NaN
4	NaN	This article needs additional citations for ve...	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...
650	NaN	NaN	The Designer	2020 (April)	DC	James Tynion IV, Jorge Jiménez	Batman Vol 3 #89
651	NaN	NaN	Vengeance	2021 (June)	DC	James Tynion IV, Guillem March	The Joker Vol 2 #2
652	NaN	NaN	Respawn	2021 (June)	DC	Joshua Williamson, Gleb Melnikov	Robin Vol 3 #1
653	NaN	NaN	Devil Ray	2021 (November)	DC	Dwayne McDuffie	Black Manta #1
654	NaN	NaN	Kira Kosov	2021 (November)	DC	James Tynion IV, Jorge Jiménez	Batman Vol 3 #112

655 rows × 7 columns

In [25]:
```
# 1. Write a script that combines the tables showing supervillain debuts fro

#cleaning of data

df_supervillan = df_supervillan.drop(columns=[0, 1])

df_supervillan = df_supervillan.dropna(axis = 0, how = 'all')
```

In [26]:
```
#make year column
launch_year(df_supervillan, str_index= 0, column= "Year Debuted")
```

In [27]:
```
df_supervillan.head(10)
```

Out[27]:

	Character / Team	Year Debuted	Company	Creator/s	First Appearance	Year
5	Ultra-Humanite	1939 (June)	DC	Jerry Siegel, Joe Shuster	Action Comics (vol. 1) #13	1939
6	Dr. Death	1939 (July)	DC	Bob Kane, Bill Finger	Detective Comics (vol. 1) #29	1939
7	The Monk	1939 (September)	DC	Bob Kane, Bill Finger	Detective Comics (vol. 1) #31	1939
8	The Claw	1939 (December)	Lev Gleason Publications	Jack Cole	Silver Streak Comics #1	1939
9	Hath-Set	1940 (January)	DC	Gardner Fox, Dennis Neville	Flash Comics #1	1940
10	Hugo Strange	1940 (February)	DC	Bob Kane, Bill Finger	Detective Comics (vol. 1) #36	1940
11	Doctor Sivana	1940 (February)	Fawcett Comics/DC	Bill Parker, C. C. Beck	Whiz Comics #2	1940
12	Lex Luthor	1940 (April)	DC	Jerry Siegel, Joe Shuster	Action Comics (vol. 1) #23	1940
13	The Joker	1940 (April)	DC	Bob Kane, Jerry Robinson, Bill Finger	Batman (vol. 1) #1	1940
14	Catwoman	1940 (April)	DC	Bob Kane, Bill Finger	Batman (vol. 1) #1	1940

In [ ]:
```
# 2. Write a script that ranks each decade in terms of how many supervillair

#First make column called decade
from math import floor
```

```
def get_decade(year):

    try:
        decade = floor(int(year) / 10) * 10
    except ValueError:
        decade = 0 #if not readable, decade will be 0, as np.nan will force

    return decade


df_supervillan["decade"] = df_supervillan["Year"].apply(get_decade)
df_supervillan = df_supervillan[df_supervillan["decade"]!= 0] #only include

df_supervillan.value_counts("decade")
```

Out[ ]:
```
decade
1960    226
1970     95
1980     89
1990     84
2000     51
1940     46
1950     26
2010     14
2020      6
1930      4
Name: count, dtype: int64
```

In [46]:
```
# 3. Write a script that ranks the different comics companies in terms of ho
df_supervillan.value_counts("Company")
```

Out[46]:
```
Company
DC                          341
Marvel                      266
Dark Horse                    6
Fawcett Comics/DC             5
Image                         5
Disney/Hyperion               4
Marvel/Timely                 4
Eternity                      3
Comico                        1
Image Comics                  1
Lev Gleason Publications      1
Mirage                        1
Name: count, dtype: int64
```

Assignment 2 –

Data analysis using correlation and regression

portfolio 2

Markus Lundsfryd Jensen

2025-12-12

```
#load packages

install.packages("pacman")
```

```
## Installing package into '/usr/local/lib/R/site-library'
## (as 'lib' is unspecified)
```

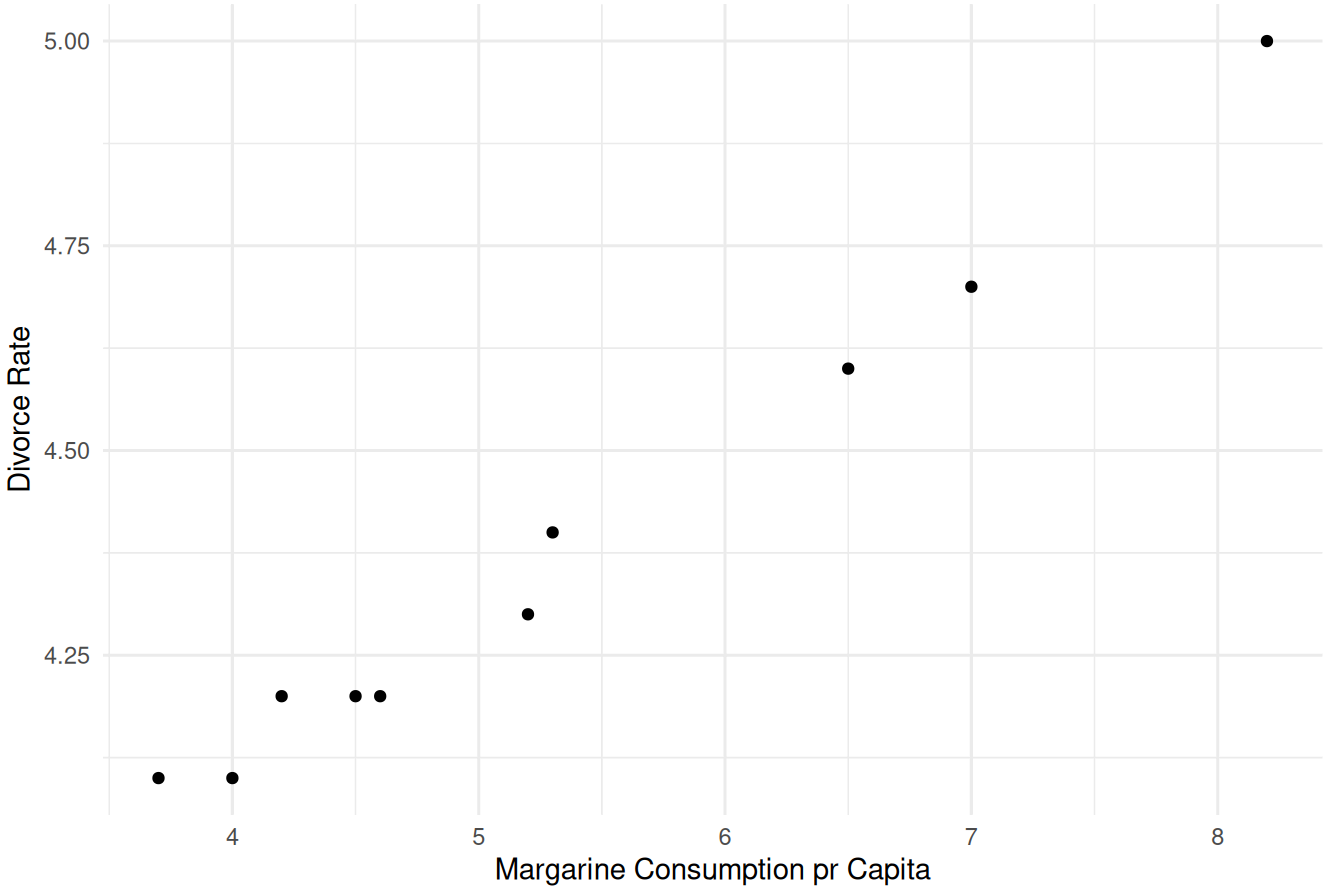
```
pacman::p_load("tidyverse", "dslabs", "car")
```

Part 1 Use the ‘divorce_margarine’ dataset from the ‘dslabs’ package to explore the relationship between margarine consumption and divorce rates in Maine. Visualise the data, make a correlation test. Report the correlation coefficient and p-value in APA format, and briefly discuss the results in a practical way.

```
data("divorce_margarine")
df = divorce_margarine
```

#relationship between magerine consuption and diverse rates. Firs a scatterplot is created

```
ggplot(df, aes(x = margarine_consumption_per_capita, y = divorce_rate_maine))+
  geom_point()+
  labs(title = "Scatter Plot of Margarine Consumption pr Capita and Divorce Rating in Maine",
       x = "Margarine Consumption pr Capita",
       y = "Divorce Rate")+
  theme_minimal()
```



There seems to be something here. Lets do a correlation test

In order to make correlation test, need to check if data is normally distributed

```
# Shapiro-Wilk normality test for margarine consumption
shapiro.test(df$margarine_consumption_per_capita)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  df$margarine_consumption_per_capita
## W = 0.90531, p-value = 0.2503
```

```
# Shapiro-Wilk normality test for Divorce rate
shapiro.test(df$divorce_rate_maine)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  df$divorce_rate_maine
## W = 0.86135, p-value = 0.07916
```

Given p values greater than 0.05 for both variables, this is interpreted as the both distributions of the variables are not significantly different from a normal distribution

```
correlation <- cor.test(df$margarine_consumption_per_capita, df$divorce_rate_maine,
                    method = "pearson")
correlation
```

```
##
##  Pearson's product-moment correlation
##
## data:  df$margarine_consumption_per_capita and df$divorce_rate_maine
## t = 23.055, df = 8, p-value = 1.33e-08
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.9676666 0.9983038
## sample estimates:
##       cor
## 0.9925585
```

A Pearson's correlation test was conducted to examine the relationship between margarine consumption per capita and the divorce rate in Maine. A strong positive correlation was found between the two variables, r(8) = .99, p < .001.

#Report

The correlation test yielded a strong and significant correlation between argarine comsumption and divorce rates in Maine. Given the available data greater amounts of margarine consumption is associated with greater divorce rates in Maine. However, correlation is not causation and while this relationship is statistically significant, the most likely reason for this relationship is some confounder not availabele in this dataset.

#Part 2

##subpart 2.0 Work with the 'GSSvocab' dataset from the 'car' package, focusing only on the year 1978 (subset that year and exclude missing values).

##subpart 2.1 Investigate how vocabulary test scores (vocab) are related to education (educ). Present the relationship in a plot, fit a regression model, and report results in APA format (coefficients, p-values, and R2), followed by a short practical interpretation.

##subpart 2.2 Extend the analysis by making a new model, where you include whether a person is native- born (nativeBorn) as a predictor. Again, visualise and model the relationship.

##subpart 2.3 Consider whether education and native-born status interact in predicting vocabulary, and if so, fit an interaction model. For each model, report results in APA format and explain the practical meaning.

##subpart 2.4 Finally, compare the models and discuss which one performs best based on p-values and R2, as well as the real-world interpretability of the results

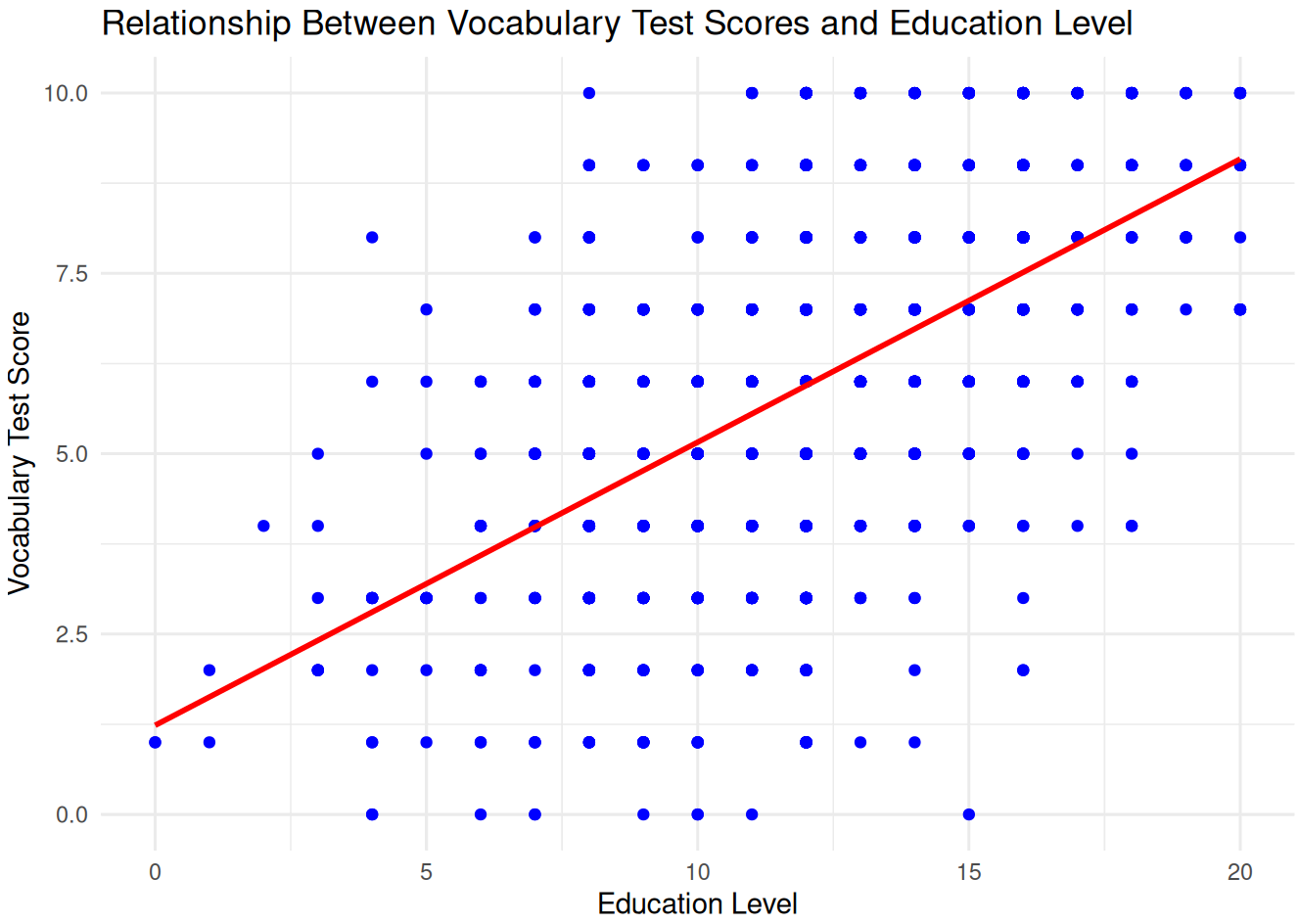
#Answers for the above mentioend subparts:

##subpart 2.0

```
#Filter for the year 1978
vocab_df <- vocab_df %>%
  filter(year == 1978)

#omit all rows containing NA's
vocab_df <- na.omit(vocab_df)
```

##subpart 2.1

```
#Make plot with regression line and datapoints
ggplot(vocab_df, aes(x = educ, y = vocab)) +
  geom_point(color='blue') +
  geom_smooth(method = "lm", formula = y ~ x, se = FALSE, color = "red")+
  labs(
    title = "Relationship Between Vocabulary Test Scores and Education Level",
    x = "Education Level",
    y = "Vocabulary Test Score"
  )+
  theme_minimal()
```




```
#Define model
vocab_edu_model <- lm(vocab ~ educ, data = vocab_df)

#Create summary statistics
summary(vocab_edu_model)
```

```
##
## Call:
## lm(formula = vocab ~ educ, data = vocab_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.1233 -1.1608  0.0542  1.0917  5.6243
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.23567    0.19957   6.192  7.7e-10 ***
## educ         0.39251    0.01606  24.443  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.885 on 1475 degrees of freedom
## Multiple R-squared:  0.2883, Adjusted R-squared:  0.2878
## F-statistic: 597.5 on 1 and 1475 DF,  p-value: < 2.2e-16
```

##Report on model The simple linear regression model fitted yields a significant positive effect of education level on vocabulary test scores, b = 0.39, t(1475) = 24.44, p< .001. The intercept was also found significant, b = 1.24, t(1475) = 6.19, p< .001. The adjusted R^2 was found to be R^2 = .29, F(1, 1475) = 597.50, p< .001.

The relationship between education level and vocabulary test scores found in this model, shows that the estimated base-test score (that is with an education level of zero) is 1.24 (the intercept). From here, an increase in test scores of .39 is estimated for each one-unit increase in education level.

##subpart 2.2

Make model

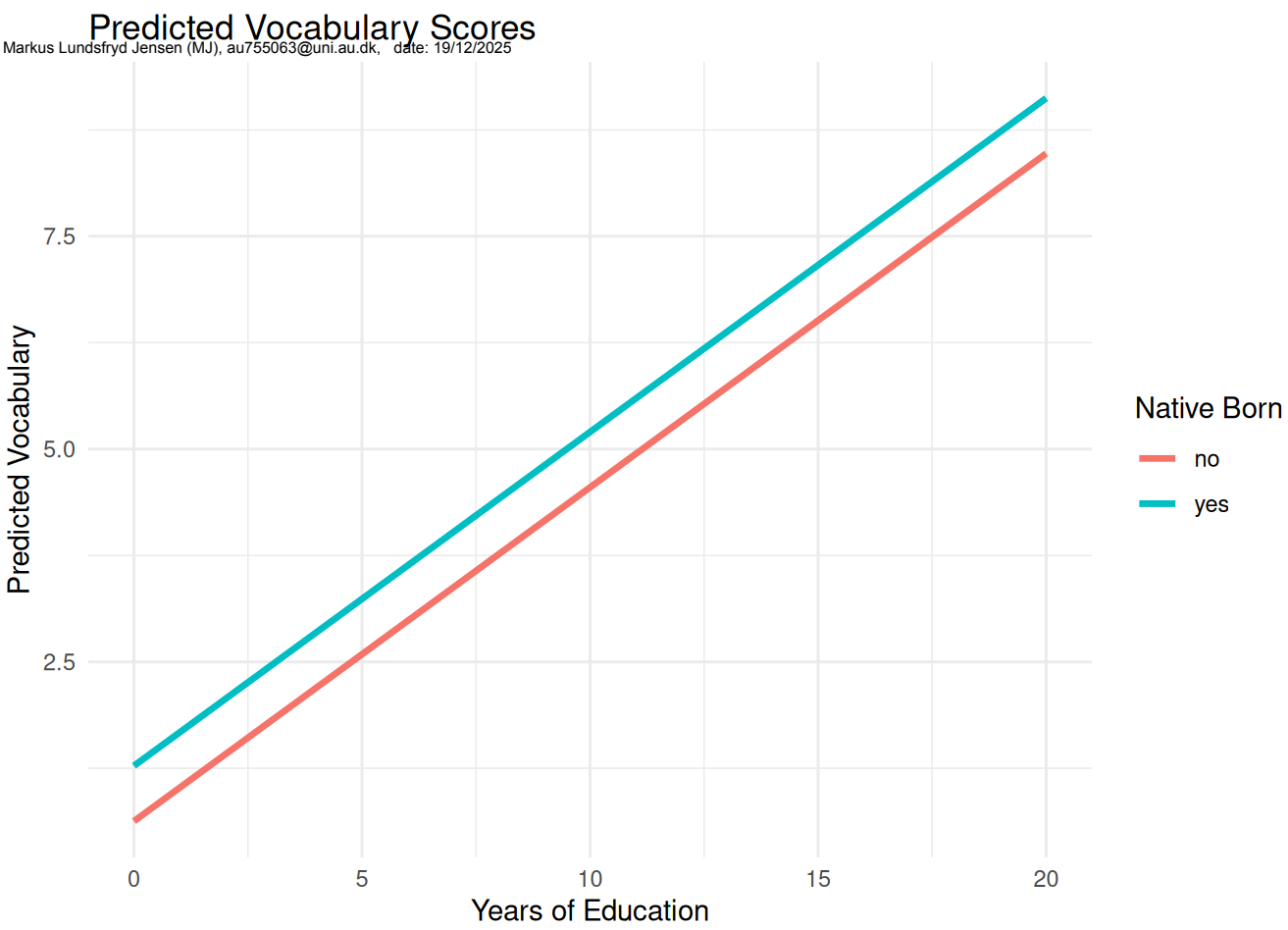
```
#Define model
vocab_edu_nat_model <- lm(vocab ~ educ + nativeBorn, data = vocab_df)

#Create summary statistics
summary(vocab_edu_nat_model)
```

```
##
## Call:
## lm(formula = vocab ~ educ + nativeBorn, data = vocab_df)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -7.162 -1.200  0.015  1.231  5.803
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.62803    0.27651   2.271  0.02327 *
## educ           0.39222    0.01601  24.499  < 2e-16 ***
## nativeBornyes  0.65032    0.20551   3.164  0.00159 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.879 on 1474 degrees of freedom
## Multiple R-squared:  0.2931, Adjusted R-squared:  0.2921
## F-statistic: 305.6 on 2 and 1474 DF,  p-value: < 2.2e-16
```

Visualize relationship with model estimates

```
# Create prediction grid
newdat <- expand.grid(
  educ = seq(min(vocab_df$educ), max(vocab_df$educ), length.out = 100),
  nativeBorn = unique(vocab_df$nativeBorn)
)

newdat$pred_vocab <- predict(vocab_edu_nat_model, newdata = newdat)

ggplot(newdat, aes(x = educ, y = pred_vocab, color = nativeBorn)) +
  geom_line(size = 1.2) +
  labs(
    title = "Predicted Vocabulary Scores",
    x = "Years of Education",
    y = "Predicted Vocabulary",
    color = "Native Born"
  ) +
  theme_minimal()
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



##Report on the model Nativeborn also proved a significant predictor of vocabulary scores with native born people being estimated to perform better than non-native born by a difference of .65 on their vocabulary test scores, b = 0.65, SE = 0.21, t(1474) = 3.16, p = .002. When controlling for native speakers, some of the variance ealier captured by the intercept now seems to be be accounted for by the native-speaker variable causing the intercept to be smaller than before, b = 0.63, t(1474) = 2.27, p = .002. The predictor education was unchanged by adding the second predictor, b = 0.39, t(1475) = 24.44, p< .001. However the adjusted R^2 is slightly greater for this model, R^2 = .29, F(1, 1474) = 305.6, p< .001.

##subpart 2.3

It might be that the relationship between vocabulary test scores and education behave differently for native and non-native speakers. Therefore an interaction model is created

```
vocab_edu_nat_int_model <- lm(vocab ~ educ + nativeBorn + educ * nativeBorn, data = vocab_df)

#Create summary statistics
summary(vocab_edu_nat_int_model)
```

```
##
## Call:
## lm(formula = vocab ~ educ + nativeBorn + educ * nativeBorn, data = vocab_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.1554 -1.2049  0.0149  1.2347  5.9857
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)         0.35394    0.68780   0.515    0.607
## educ                0.41510    0.05496   7.553 7.45e-14 ***
## nativeBornyes       0.95000    0.71855   1.322    0.186
## educ:nativeBornyes -0.02501    0.05745  -0.435    0.663
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.88 on 1473 degrees of freedom
## Multiple R-squared:  0.2932, Adjusted R-squared:  0.2917
## F-statistic: 203.7 on 3 and 1473 DF,  p-value: < 2.2e-16
```

##Report on model While the interaction term tells an interesting story of a slightly negative estimate of education for native born people (meaning an increase in education is affecting vocabulary in a negative direction compared to non-native speakers), this interaction is far from significant, b = -0.03, t(1473) = -0.45, p > .05. Education level was found a positive and significant predictor, b = 0.42, t(1473) = 7.55, p < .001. However, including the interaction term caused the predictor of NativeBorn to not be significant, b = 0.95, t(1473) = 1.32, p > .05. The change in adjusted R^2 was negligible compared to the former model, R^2 = .29, F(1, 1473) = 203.7, p< .001.

##subpart 2.4

```
anova(vocab_edu_model, vocab_edu_nat_model, vocab_edu_nat_int_model)
```

```
## Analysis of Variance Table
##
## Model 1: vocab ~ educ
## Model 2: vocab ~ educ + nativeBorn
## Model 3: vocab ~ educ + nativeBorn + educ * nativeBorn
##   Res.Df    RSS Df Sum of Sq       F  Pr(>F)
## 1   1475 5241.8
## 2   1474 5206.5  1    35.371 10.0083 0.00159 **
## 3   1473 5205.8  1     0.670  0.1894 0.66344
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

##Report on modelcomparison Likelihood ratio test using anova was conducted comparing the created models of increasing complexity to see if the increase in complexity would be justified by a better model fit. Here, the linear model predicting vocabulary test scores by education level and native speaker, proved significantly better than the model only holding education level, X^2(1) = 10.0, p = .0016. Adding an interaction term did not significantly improve model fit, X^2(1) = 0.19, p > 0.05. In addition to the anova performed, inspection of R^2 also suggest education level and native speaker model as the better fit with a slightly greater R^2. However, this difference is close to negliable.

Thus, it seems that while education level is a good predictor of vocabulary, whether or not the person being evaluated is being tested in his/her mother toungue also has a great effect on the estimated scores of the person in question. At least when evaluating data from 1978

Assignment 3 –

Analysis of own project data

Rune Egeskov Trust (RT), au692561@uni.au.dk

Markus Lundsfryd Jensen (MJ), au755063@uni.au.dk

Assignment 3: Airbnb NY Data Analysis

2025-11-05

```
data <- read_csv("/work/CulturalDataScience/NY_data_with_sent.csv",show_col_types = FALSE)
```

```
## New names:
## • `` -> `...1`
```

```
# Rune
# Setting room type to factor to allow for leveled analysis
data$room_type <- as.factor(data$room_type)
# Inspecting data types to see if anything should be coerced
spec(data)
```

```
## cols(
##   ...1 = col_double(),
##   id = col_double(),
##   name = col_character(),
##   host_id = col_double(),
##   host_name = col_character(),
##   neighbourhood_group = col_character(),
##   neighbourhood = col_character(),
##   latitude = col_double(),
##   longitude = col_double(),
##   room_type = col_character(),
##   price = col_double(),
##   minimum_nights = col_double(),
##   number_of_reviews = col_double(),
##   last_review = col_date(format = ""),
##   reviews_per_month = col_double(),
##   calculated_host_listings_count = col_double(),
##   availability_365 = col_double(),
##   vs_neg = col_double(),
##   vs_pos = col_double(),
##   vs_neu = col_double(),
##   vs_compound = col_double(),
##   sentiment = col_character()
## )
```

```
# Markus
# Omitting zero pricing (uninformative and creates NAs when log transformed)
data <- data %>% filter(price>0)
```

```
# Rune
# Omitting NAs in the name column (creates NAs when passed to VADER)
data <- data[!is.na(data$name), ]
# Log transforming price variable and saving to new column
data$log_price <- log(data$price)
```

```
# Markus
# Selecting relevant columns
data <- data %>% select(price, name,vs_compound,log_price, neighbourhood, room_type)
```

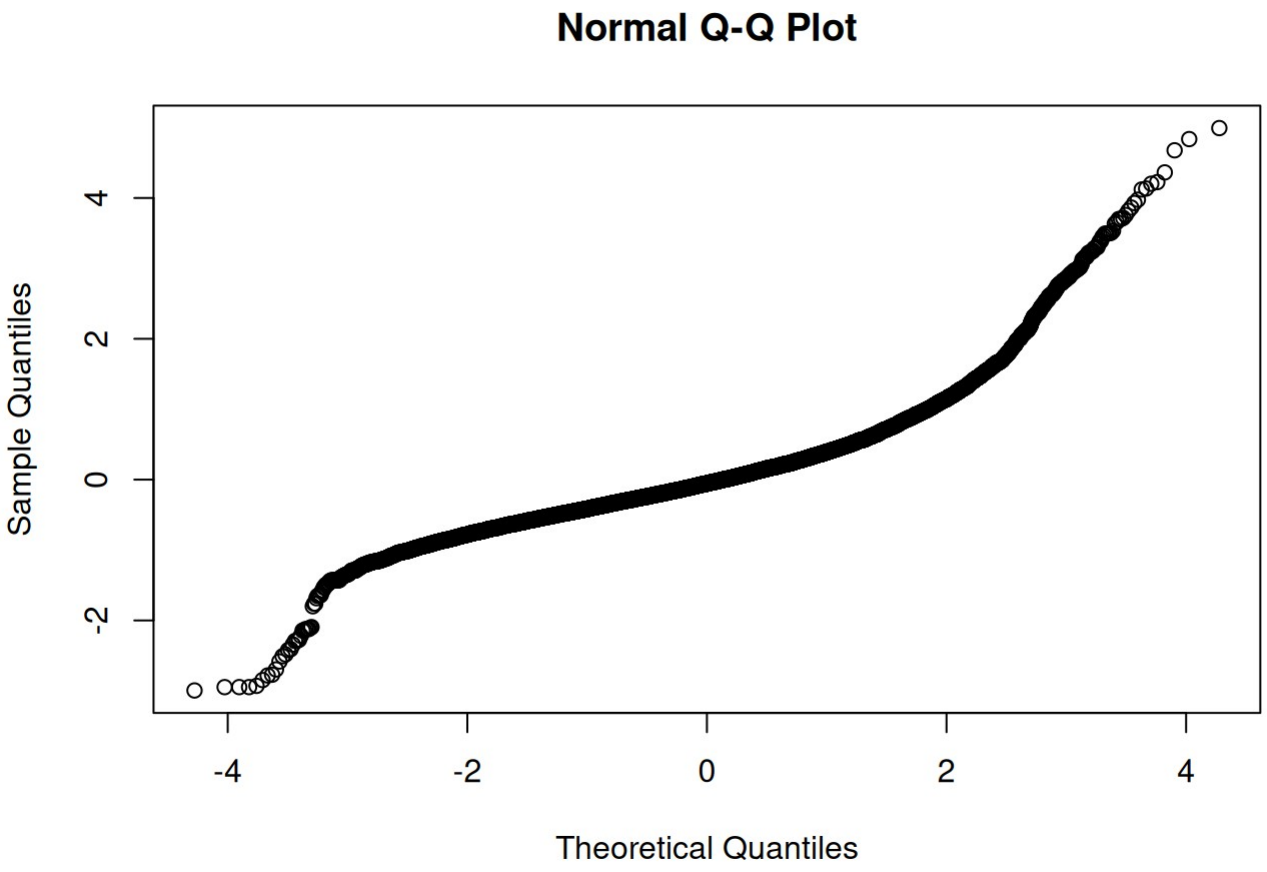
```
# Rune
# Creating a mixed effects model that allows for a random intercept for neighbourhood and room type (who
le aparment or only room) as these intuitively carry price information
model_sent_neighbor <- lmerTest::lmer(log_price ~ vs_compound + (1|neighbourhood) + (1|room_type), data=
data)

model_sent_neighbor_no_log <- lmerTest::lmer(price ~ vs_compound + (1|neighbourhood) + (1|room_type), da
ta=data)

# From the summary we see a statistically significant negative effect of sentiment on the price per nigh
t
summary(model_sent_neighbor)
```

```
## Linear mixed model fit by REML. t-tests use Satterthwaite's method [
## lmerModLmerTest]
## Formula: log_price ~ vs_compound + (1 | neighbourhood) + (1 | room_type)
##    Data: data
##
## REML criterion at convergence: 73622.4
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -6.2074 -0.6177 -0.1137  0.4675 10.3482
##
## Random effects:
##  Groups        Name        Variance Std.Dev.
##  neighbourhood (Intercept) 0.08658  0.2942
##  room_type     (Intercept) 0.29138  0.5398
##  Residual                  0.23283  0.4825
## Number of obs: 52838, groups:  neighbourhood, 221; room_type, 3
##
## Fixed effects:
##               Estimate Std. Error         df t value Pr(>|t|)
## (Intercept)  4.340e+00  3.124e-01  2.016e+00  13.891  0.00499 **
## vs_compound -6.083e-02  7.096e-03  5.269e+04  -8.573  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr)
## vs_compound -0.005
```

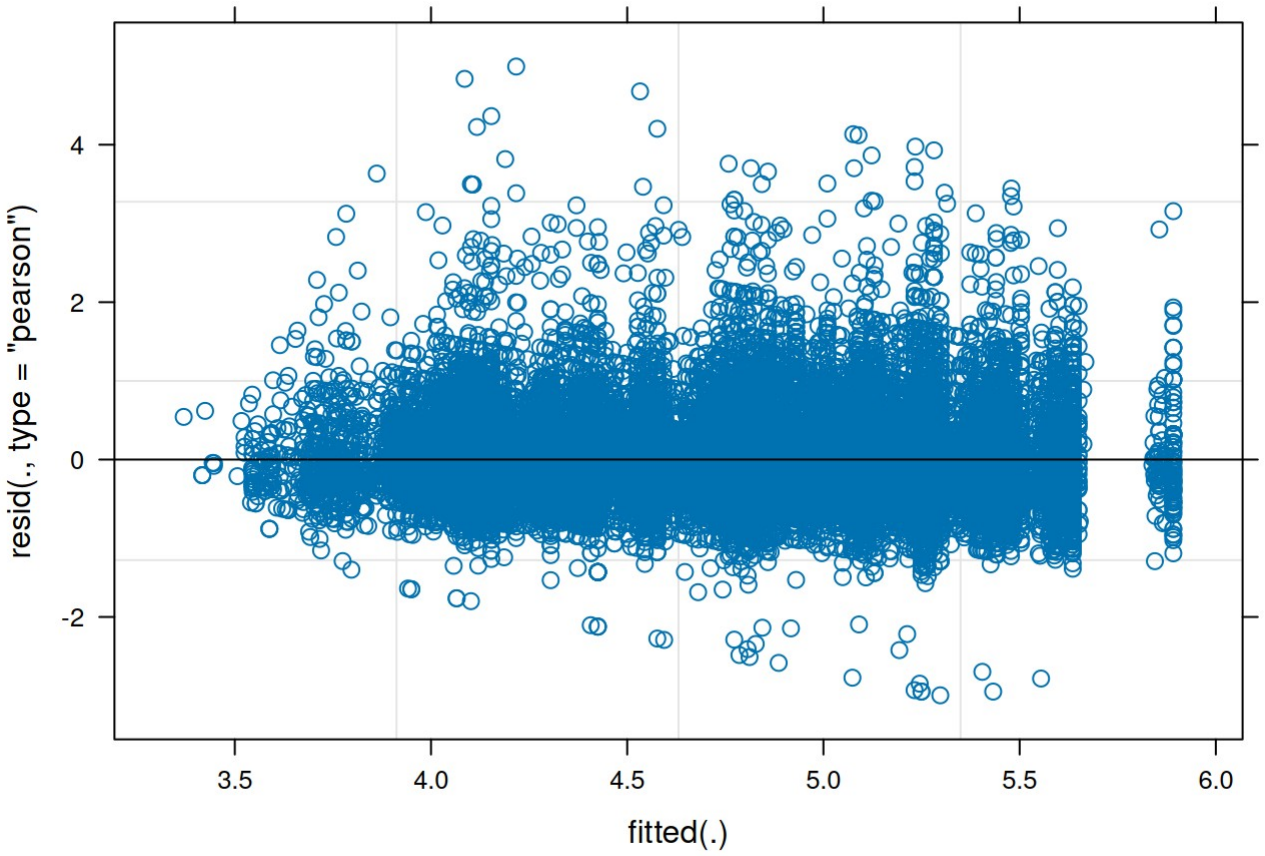
```
# We see an adjusted r-squared value of .000542
r.squaredGLMM(model_sent_neighbor)
```

```
##              R2m       R2c
## [1,] 0.000543319 0.6190085
```

```
# Markus
# Log transformation of price variable has moved the residuals closer to the theoretical line for normal
ity of residuals, but still does not meet the assumptions - we will touch on this in limitations.
qqnorm(residuals(model_sent_neighbor))
```




```
# Rune
plot(model_sent_neighbor)
```




```
# Markus
# Plotting the model for a visual representation of the observed effect.
plot_model(model_sent_neighbor, type="pred", terms="vs_compound",
           axis.title = c("Vader Sentiment Compound Score", "Logarithm of price per night in USD"),
           title = "Predicted Logarithmic Price by Compound Score",
           ci.lvl=NA) +
  theme_minimal()
```


Predicted Logarithmic Price by Compound Score
Logarithm of price per night in USD
4.400
4.375
4.350
4.325
4.300
4.275
-1.0
-0.5
0.0
0.5
1.0
Vader Sentiment Compound Score

Assignment 4 –

In class project presentation

Rune Egeskov Trust (RT), au692561@uni.au.dk

Markus Lundsfryd Jensen (MJ), au755063@uni.au.dk

How Much Is 'Beautiful' Worth?

Sentiment-Driven Pricing in NYC Airbnb Listings

Rune and Markus

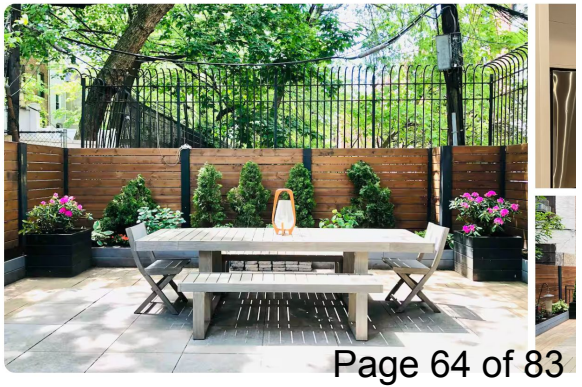
Introduction 1.0, AirBnb Pricing

- Abundance of choices, many factors influence decision
- Price - Demand - Proxy for WTP
- Attractions, transportation, amenities (70-80%) (List, 2025)

- Less visible relationship - Daily rates of Airbnbs in New York City and the sentiment of the titles.
- “Beautiful BRIGHT and SUNNY - BEST location in NYC!

Sunny and Beautiful 1 Bedroom in Brooklyn

Lovely room w/king bed in shared garden duplex.



Introduction 1.1, Sentiment Analysis

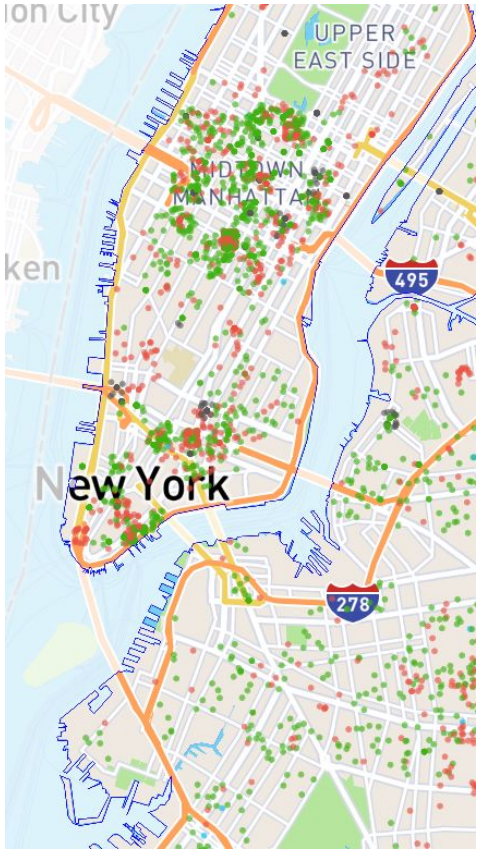
- Sentiment analysis
  - Subdomain of NLP
  - Emotional tone, Negative, Neutral, Positive
- Sentiment and consumer behavior
  - Product descriptions and WTP, increased perceived value (Ijaz et al., 2025)
  - AirBnB reviews, positive sentiment -> higher price tag (Lawani et al., 2019)
  - However, positive claims ->
    - Increase consumer scepticism ->
      - Focus on the flaws ->
        - Chance of disappointment (Ford et al., 1990)
  - Additionally, positive expectations -> higher satisfaction,
    - but disconfirmation -> amplifies the negatives, == dissatisfaction (Schiebler et al., 2025)

Research Question and Hypothesis

- So, In this paper, relationship between price and sentiment examined
- Possible effect is would not be a direct one, Altered expectancy
- RQ: How does the specific title of an AirBnB listing influence the price per night?
- Null hypothesis
    - The title used for a listing does not influence pricing of the given listing.
- Alternative hypothesis
    - The title used for a listing has an influence on pricing.

2.1 Methods

- Data sourcing
  - Kaggle (Open license):
    - Originally sourced from “Inside AirBnB” project
- Data cleaning / analyzing
  - Data was cleaned before being submitted to Kaggle
  - Further analysis:
    - Sentiment using VADER
    - Adding sentiment columns
- Compound score:
  - ‘normalized, weighted composite score’ for sentiment
  - Between -1 and 1
  - -1 is fully negative, and 1 fully positive



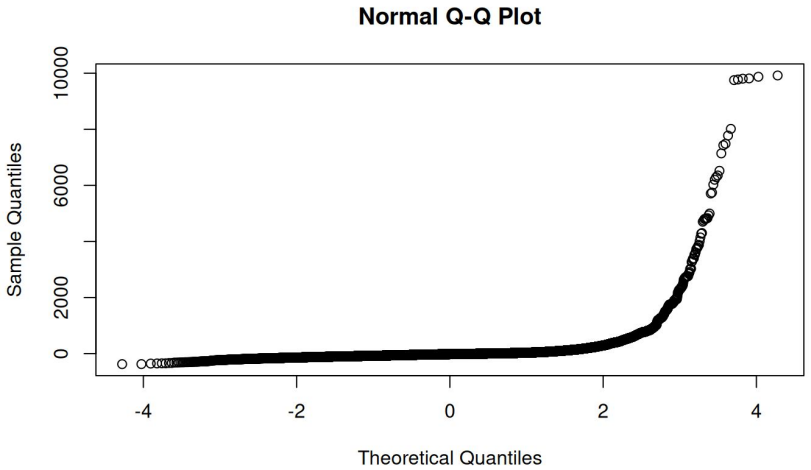
Airbnb properties in NYC

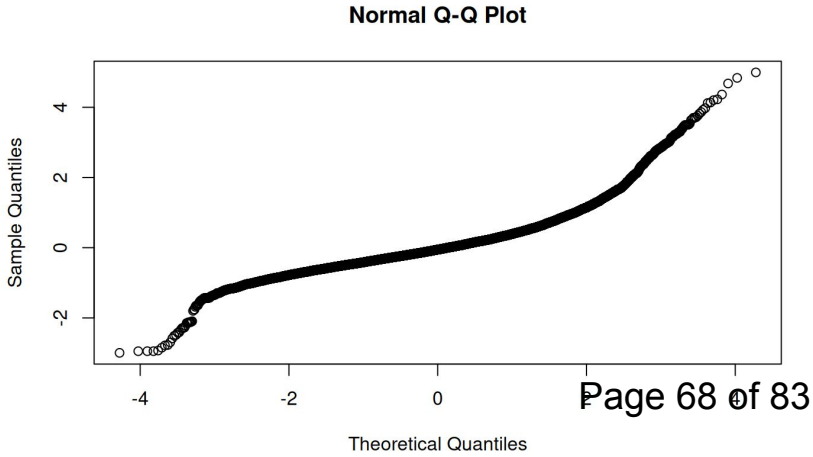
2.2 Methods

- Linear mixed effects regression
  - Random intercept for “room type” and “neighbourhood”
    - Taking variance from these variables into account
    - Only 1 data point per listing = no random slope
  - Log price by “compound score”

Price ~ Compound + (1 | Neighbourhood) + (1 | Room type)

- Non-normal residuals required scaling
  - Top: Standard scale
  - Bottom: Log-transformed
  - Closer to meeting assumption





Results

- Significant intercept
  - Intercept = 4.34
  - p = .005
- Significant slope
  - Price ↓ if sentiment ↑
  - Effect size = -.0607
  - p < .001
- Adjusted r-squared = .00053

Discussion, Interpretation 1

- Positive sentiment -> lower prices
- Small explainability of variance

- 1) Creates heightened expectations ->
    - increase consumer scepticism ->
        - observant of any flaws ->
            - fewer bookings ->
                - Lower prices

Discussion, Interpretation 2

- Heightened expectations are not initially disconfirmed ->
  - Disconfirmation happening during the stay ->
    - Bad review(s) ->
      - Fewer future bookings ->
        - Lower prices

Me: *about to buy a highly rated product I've been researching for days*

sees literally 1 bad review

Me:

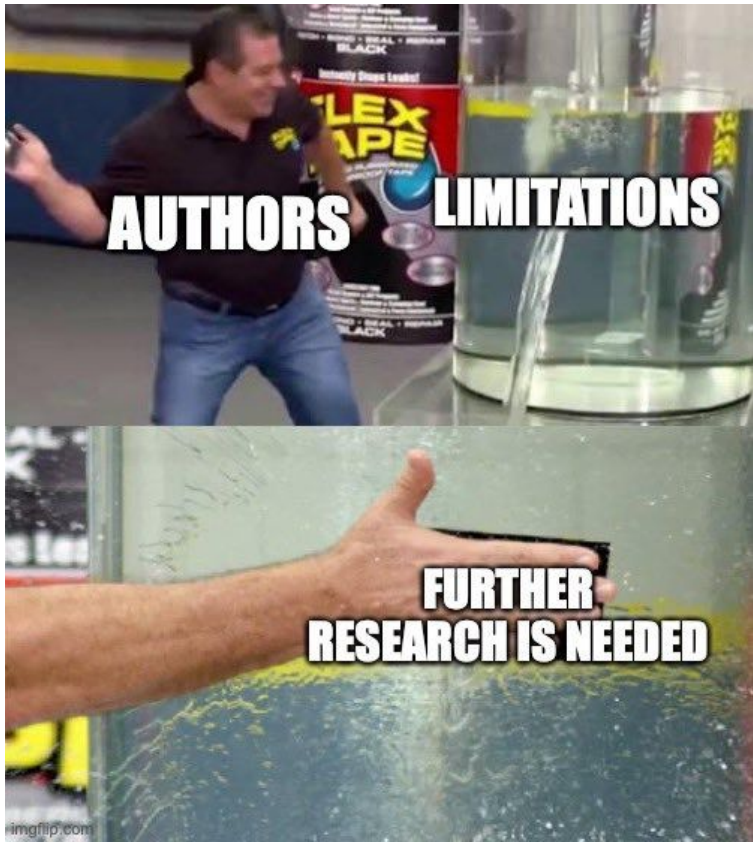
Limitations

Limitations:

- Text amount (short titles)
  - Less data = worse result
- Linear model assumptions
  - Non-normal distribution
- R-squared
  - Low variance explained
  - Much is not captured by model

Future Directions:

- Sentiment of full description
- Looking at influence on reviews
- Titles more positive if bad reviews?



Literature

Ford, G. T., Smith, D. B., & Swasy, J. L. (1990). Consumer Skepticism of Advertising Claims: Testing Hypotheses from Economics of Information. *Journal of Consumer Research*, *16*(4), 433–441. https://doi.org/10.1086/209228

Hutto, C. J., & Gilbert, E. E. (2014, June). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. *Proceedings of the Eighth International Conference on Weblogs and Social Media (ICWSM-14)*.

Ijaz, A., Yu, J., Schwab, B., & Cho, J. (2025). Quality signaling and willingness-to-pay: An experimental assessment. *Marketing Letters*, *36*(3), 341–353. https://doi.org/10.1007/s11002-025-09769-3

Lawani, A., Reed, M. R., Mark, T., & Zheng, Y. (2019). Reviews and price on online platforms: Evidence from sentiment analysis of Airbnb reviews in Boston. *Regional Science and Urban Economics*, *75*, 22–34. https://doi.org/10.1016/j.regsciurbeco.2018.11.003

Mao, Y., Liu, Q., & Zhang, Y. (2024). Sentiment analysis methods, applications, and challenges: A systematic literature review. *Journal of King Saud University - Computer and Information Sciences*, *36*(4), 102048. https://doi.org/10.1016/j.jksuci.2024.102048

Schiebler, T., Lee, N., & Brodbeck, F. C. (2025). Expectancy-disconfirmation and consumer satisfaction: A meta-analysis. *Journal of the Academy of Marketing Science*. https://doi.org/10.1007/s11747-024-01078-x

Sharma, H. D., & Goyal, P. (2023). An Analysis of Sentiment: Methods, Applications, and Challenges. *Engineering Proceedings*, *59*(1), 68. https://doi.org/10.3390/engproc2023059068

List, M. (2025). *U.S. Short-Term Rentals 2025: Guest Attitudes and Decision Making: Phocuswright*. https://www.phocuswright.com/Travel-Research/Consumer-Trends/US-Short-Term-Rentals-2025-Guest-Attitudes-and-Decision-Making

Assignment 5 –

Short written assignment on own project

Rune Egeskov Trust (RT), au692561@uni.au.dk

Markus Lundsfryd Jensen (MJ), au755063@uni.au.dk

How Much Is 'Beautiful' Worth?

Sentiment-Driven Pricing in NYC Airbnb Listings

Rune Egeskov Trust (RT), au692561@uni.au.dk
Markus Lundsfryd Jensen (MJ), au755063@uni.au.dk

School of Communication and Culture, Aarhus University
Jens Chr. Skous Vej 2, 8000 Aarhus C, Denmark
Supervisor: Anna Zamm

1. Introduction (MJ)

“Beautiful BRIGHT and SUNNY - BEST location in NYC!” is one of the many colorful titles of the Airbnbs in New York City. When consumers are searching for accommodation on platforms such as Airbnb, they are met with an abundance of choices. Many factors are at play when choosing where to stay, and how much one is willing to pay for it. Under the assumption that prices of Airbnbs are strictly affected by the demand of consumers, price can be used as a proxy of the consumers’ willingness to pay (WTP). Often, such WTP is decided by quality, including the available amenities and location, in association with the price of the establishment in question. A comprehensive consumer research study from 2025 found the top drivers for choosing short-term accommodation rentals to be proximity to attractions, options regarding transportation and amenities with geographic accessibility influencing 70-80% of decisions in studies (List, 2025).

This paper, however, moves to assess a less prominent relationship given by the daily rates of Airbnbs in New York City and the sentiment of the titles describing the accommodations.

1.1 Sentiment analysis (MJ)

Sentiment analysis is a subdomain within Natural Language Processing, used to determine the prevalent emotional tone in the provided text and classify such as either negative, neutral or positive. With this tool, it has become possible to efficiently identify textualized sentiment on enormous amounts of data (Mao et al., 2024). This method of inquiry has been put to use in a wide variety of applications for businesses, governments, and organizations in addition to a wide bouquet of research topics (Sharma & Goyal, 2023).

1.2 Sentiment and Consumer Behavior (MJ)

To examine the connection between sentiment of text connected to a product and the indirect connection with price through changes in consumer behavior, is not a new concept. One study investigated product descriptions and found increased WTP by an increase in perceived value when the product descriptions had a positive sentiment (Ijaz et al., 2025). More specifically, Airbnb itself has been subject to research investigating the influence of sentiment in reviews and the associated price of the accommodation. One such study found reviews to serve as a quality metric and influence consumers' purchasing decisions in a way

where positive sentiment of reviews was associated with a higher price tag (Lawani et al., 2019).

While positive sentiment until now has been put forth as a positive driver of price, another aspect is also in need of introduction. In some cases, positive claims will increase consumer scepticism causing the consumer to focus on the flaws of the establishment (Ford et al., 1990). With consumers' focus shifted to the negative, the chance of being disappointed is also increased, resulting in either fewer bookings directly or in worse reviews which again will drive down the demand. Similar findings have been found elsewhere, with positive expectations correlating with higher satisfaction, but any disconfirmation of these expectations amplifies the negatives, leading to dissatisfaction (Schiebler et al., 2025).

In this paper, sentiment will be measured on the titles of Airbnbs, and its relationship with price will be looked into. The possible effect of sentiment of price, however, is not suspected to be a direct one. Such an effect is most likely derived as a subproduct of altered expectancy due to changes in sentiment increasing the chances for higher satisfaction, or if disconfirmed, amplified dissatisfaction and thus worse reviews. This examination goes to tell a story about how humans react to listings of varying sentiment. Thus, the research question formulated for this inquiry has been outlined beneath:

RQ: How does the sentiment of an Airbnb listing title influence the price per night, and what possible explanations of changes in consumer behavior can be inferred from this?

2. Methods (RT)

Data was sourced from Kaggle, which contained a cleaned version of data from the open Inside Airbnb project (https://insideairbnb.com/new-york-city/). The dataset contains information about each active Airbnb listing in New York City in 2018, for a total of 52,838 observations. For the purpose of examining the influence of sentiment of the listing title on the price per night of the given listing, the Python package VADER was used for the actual sentiment analysis (Hutto & Gilbert, 2014).

The operationalisation of sentiment happened by extracting the so-called compound score from VADER for each listing. This score is defined as a “normalized, weighted composite

score" measured on a scale between -1 and 1. This in practice means that it can be used if only one number is wanted to describe the sentiment of the input text. For context, scores below -0.5 can be categorized as negative, scores between -0.5 and 0.5 as neutral, and finally scores above 0.5 as positive.

In order to examine the relationship, a linear mixed effects model was used to predict price per night by the compound score of the listing title. As it is known that location has an influence on the price of the accommodation, a random intercept for neighbourhood was used. As the dataset also contained information about the room type, if it was an entire apartment or simply a room, a random intercept was also used to take the difference in price between these types into account.

Looking at the QQ-plot when checking for normality of residuals, it became quite clear that the non-transformed data did not fulfill this assumption for the linear model, as residuals were skewed. In order to mediate this issue, the price per night variable was log transformed. This reduced skew to an acceptable, if not perfect, level, and the linear model was implemented with the logarithmic price per night as the dependent variable.

3. Results (RT)

A linear mixed effects model was fit to the data, which suggested a statistically significant downward trend in price per night as the compound score rose toward 1; b = -0.06, SE = 0.01, t(52,689) = -8.57, p < .001. The intercept was also significant; b = 4.34, SE = 0.31, t(2.02) = 13.89, p = .005 The model had an adjusted r-squared value of .00053. For a visual representation of the trend, see figure 1.

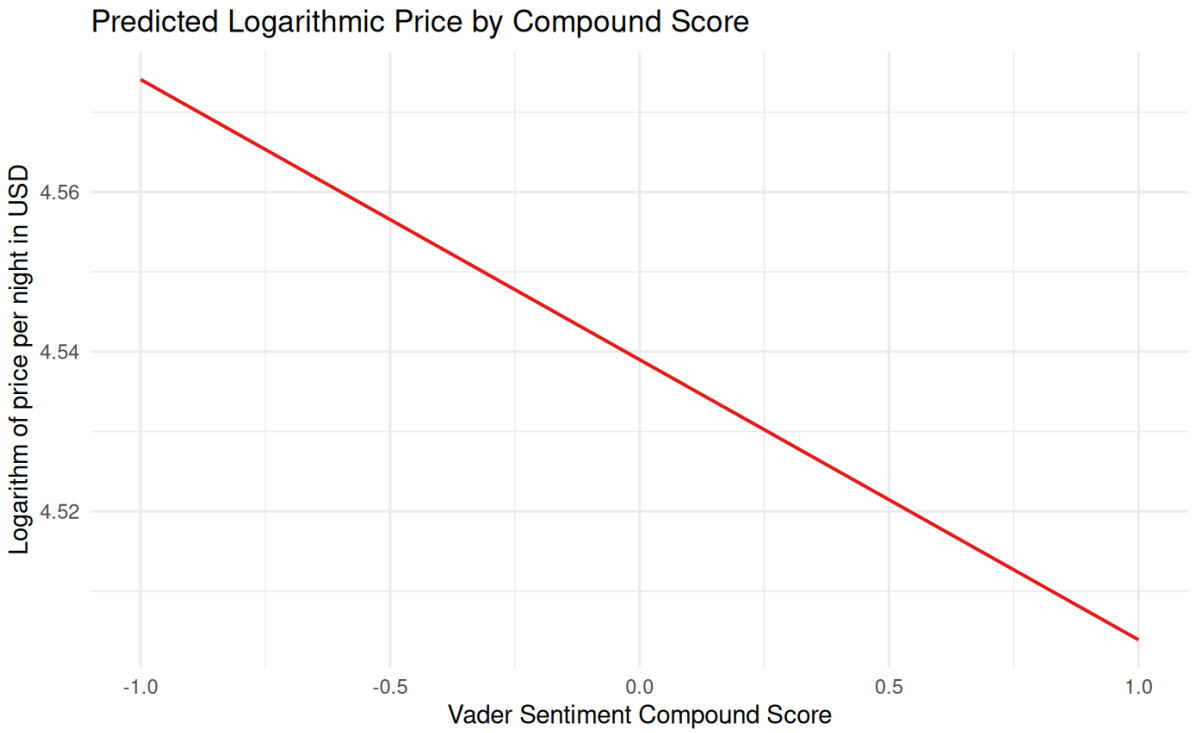


Figure 1: Visual representation of relationship between price and compound score

4. Discussion

4.1 Interpretation of the Results (MJ)

During analysis, a negative relationship was found between sentiment score of title listings and the predicted price. This suggests that as the listings become increasingly positive the prices of Airbnb's in NYC decline. This is most likely a byproduct of multiple processes happening simultaneously and in varying directions making an adequate causal interpretation unfeasible. However, the authors have proposed two partial explanations of the underlying cause of change in price associated with the findings of this paper.

First, it might be that increasingly positive sentiment of titles, which is the first thing that consumers meet when examining the accommodation, creates heightened expectations regarding this establishment. As outlined in the introduction section, such positivity might also increase consumer scepticism making the consumer more observant of any flaws prevalent on the listing itself that counters the initial expectations made about the place, which, when disconfirmed, amplifies negatives. This will result in fewer bookings, as

possible customers opt for a different accommodation, where expectations are made when pressing the link, matches the pictures, descriptions and specific reviews of the establishment.

Another equally likely option is that the heightened expectations are not initially disconfirmed resulting in such disconfirmation happening during the stay. While this does not immediately influence demand, any bad review left after such stay will. The concrete sequence of events has not been the primary focus of this paper and future studies are urged to further examine such underlying causes of the effect found in this paper.

It is worth noticing that while the effect size is non-negliable and very significant, the effect found in this paper might be the cause of some underlying confounder. This will be discussed further in the following section.

4.2 Limitations and Future Directions (RT)

There are multiple limitations of the present study, ranging from statistical to methodological. In the methodological realm is the implementation of VADER. While VADER may be one of the better sentiment analysis tools available (Mao et al., 2024), it still does not have much text to work with, as this paper exclusively looks at the short titles of listings.

In the statistical realm there are a number of limitations. The assumption of normal distribution of residuals was not met fully, even after log-transformation of the price variable. When assumptions are not met, it means that the inferences drawn from the model can be shaky. Another limitation is the admittedly low r-squared value, which tells us that the linear prediction accounts for a very small amount of total variance - indicating that a lot of factors are unaccounted for in the model.

Going forward, it would be interesting to look into whether a discrepancy exists between reviews and sentiment score of the title - specifically, if a relationship can be found that links a higher sentiment score to worse reviews.

5. Literature

Ford, G. T., Smith, D. B., & Swasy, J. L. (1990). Consumer Skepticism of Advertising Claims: Testing Hypotheses from Economics of Information. *Journal of Consumer Research*, *16*(4), 433–441. https://doi.org/10.1086/209228

Hutto, C. J., & Gilbert, E. E. (2014, June). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. *Proceedings of the Eighth International Conference on Weblogs and Social Media (ICWSM-14)*.

Ijaz, A., Yu, J., Schwab, B., & Cho, J. (2025). Quality signaling and willingness-to-pay: An experimental assessment. *Marketing Letters*, *36*(3), 341–353. https://doi.org/10.1007/s11002-025-09769-3

Lawani, A., Reed, M. R., Mark, T., & Zheng, Y. (2019). Reviews and price on online platforms: Evidence from sentiment analysis of Airbnb reviews in Boston. *Regional Science and Urban Economics*, *75*, 22–34. https://doi.org/10.1016/j.regsciurbeco.2018.11.003

List, M. (2025). *U.S. Short-Term Rentals 2025: Guest Attitudes and Decision Making: Phocuswright*. https://www.phocuswright.com/Travel-Research/Consumer-Trends/US-Short-Term-Rentals-2025-Guest-Attitudes-and-Decision-Making

Mao, Y., Liu, Q., & Zhang, Y. (2024). Sentiment analysis methods, applications, and challenges: A systematic literature review. *Journal of King Saud University - Computer and Information Sciences*, *36*(4), 102048. https://doi.org/10.1016/j.jksuci.2024.102048

Schiebler, T., Lee, N., & Brodbeck, F. C. (2025). Expectancy-disconfirmation and consumer satisfaction: A meta-analysis. *Journal of the Academy of Marketing Science*. https://doi.org/10.1007/s11747-024-01078-x

Sharma, H. D., & Goyal, P. (2023). An Analysis of Sentiment: Methods, Applications, and Challenges. *Engineering Proceedings*, *59*(1), 68. https://doi.org/10.3390/engproc2023059068