

Data Science Across Disciplines

Cagatay Turkay

Carlos Cámara

James Tripp

2023-08-14

Table of contents

Welcome	8
What this module is about?	8
Course contents	9
Introduction and historical perspectives (Chapter 1)	9
Thinking Data: Theoretical and Practical Concerns	9
.	9
Acknowledgements	9
Licence	9
I About	10
Teaching Staff	11
Former staff	11
II Introduction and Historical Perspectives	13
1 Session 1	14
2 IM939 - Lab 1 - Part 1	15
2.1 Python?	15
2.1.1 Anaconda	15
2.1.2 Jupyter Notebooks	16
2.2 Getting started	16
3 Data Types	17
3.1 int, floats, strings	17
3.1.1 lists and dictionaries	18
4 Variables	19
4.1 Bringing it all together	20
5 Congratulations	21
6 IM939 - Lab 1 - Part 2	22
6.1 Functions	22

6.2 Methods	23
7 Flow control	25
7.1 For loops	25
7.2 If statements	26
8 Modules	29
8.1 CSV module	29
9 Pandas	34
III Thinking Data: Theoretical and Practical Concerns	37
10 IM939 Lab 2 - Part 1	38
10.1 Help!	38
10.2 Structure	39
10.3 Summary	42
10.4 Selecting subsets	44
10.5 Adding columns	50
10.6 Writing data	51
11 IM939 Lab 2 - Part 2	52
11.1 Plots	52
11.2 Univariate - a single variable	52
11.3 Bivariate	58
11.4 Dates	59
11.5 Multivariate	62
12 IM939 Lab 2 - Part 3	67
12.1 Missing values	67
12.2 Transformations	70
13 IM939 Lab 2 - Part 4	78
13.1 Seaborn	78
13.2 Functions	80
IV Abstractions & Models	84
14 Lab: Data Processing and Summarization	85
14.1 Part I: Outliers	85
14.2 Part II : Q-Q Plots	91
14.3 Part III : Distributions, Sampling	96

14.4 Part IV : Robust Statistics	100
15 IM939 - Lab 3 - Regression	106
15.1 Scikit Learn	106
15.2 Sea Ice Dataset	106
15.3 Simple and Multiple Linear Regression	106
15.4 Ordinary Least Squares	107
16 Case study: Climate Change and Sea Ice Extent	108
16.1 Reading Data	108
16.2 Data visualisation to explore data	109
16.3 Normalization	112
16.4 Pearson's correlation	117
16.5 Simple OLS	117
17 IM939 - Lab 3 - Regression Exercise	122
17.1 Scikit Learn	122
17.2 Wine Dataset	122
17.3 Reading Data	122
17.4 Data exploration	122
17.4.1 Check your variables	123
V Structures & Spaces	125
18 IM939 Lab 4 - Part 1 - Iris	126
18.1 Data	126
18.2 Questions	128
18.3 Initial exploration	128
18.4 Clustering	132
18.5 Number of clusters	140
18.6 Principal Component Analysis (PCA)	143
18.6.1 Dimension reduction	144
18.7 PCA to Clusters	151
18.8 Missing values	154
18.8.1 Zeroing	155
18.8.2 Replacing with the average	159
19 Useful resources	162
20 IM939 Lab 4 - Part 2 - Crime	163
21 IM939 Lab 4 - Part 3 - Exercises	173
21.1 Load data and import libraries	174

22 Dimension reduction	175
22.1 SparcePCA	175
22.2 tSNE	176
22.3 Clustering	178
VI Multi-model thinking and rigour	180
23 IM939 Lab 5 - Part 1	181
23.1 Workflow	181
23.2 So far	181
23.3 This week	182
23.3.1 Clustering and ground truth	182
24 IM939 Lab 5 - Part 2	196
25 IM939 Lab 5 - Part 3	205
25.0.1 A small TODO for you:	219
26 IM939 lab 5 - Exercise	220
VII Recognising and avoiding traps	222
27 IM939 - Lab 6 - Part 1 - Setup and Illusions	223
27.1 Clustering illusion	223
28 Weber-Fechner Law	228
29 IM939 - Lab 6 - Part 2 - Axes manipulation	233
29.1 Data wrangling	233
30 IM939 - Lab 6 - Part 3 - Choropleths	242
31 IM939 - Lab 6 - Part 4 - Exercise	249
32 IM939 - Lab 6 Part 5 Simpson's Paradox	250
32.1 1 Data	250
32.1.1 Source:	250
32.1.2 Variable Descriptions:	250
32.1.3 Research problem	251
32.2 2 Reading the dataset	251
32.3 3 Exploring data	252
32.3.1 Missing values	252
32.3.2 Data types	255

32.3.3 Age	256
32.4 4 Exploratory analysis	259
32.4.1 Age x expenditures	260
32.4.2 Ethnicity	261
32.4.3 Gender	263
32.4.4 Mean Expenditures	264
33 5 In-depth Analysis - Outliers	267
34 6 Conclusions	273
34.1 Explanation	273
34.2 Takeaways	273
34.2.1 *Additional Links	274
VIII Data Science & Society	275
35 IM939 - Lab 7 Part 1	276
35.0.1 Datasets	276
35.0.2 Further datasets	276
35.0.3 Additional Readings	277
35.1 1 Hate Crimes	277
35.1.1 Source:	277
35.1.2 Variables:	277
35.2 2 Reading the dataset	278
35.3 3 Exploring data	279
35.3.1 Missing values	279
35.4 Mapping hate crime across the USA	281
35.5 Exploring data	282
36 IM939 - Lab 7 Part 2	298
36.1 Source	298
36.1.1 Definitions	298
36.1.2 Questions	299
36.2 Reading the dataset	299
36.2.1 Missing values	299
36.3 How quickly are populations growing?	307
References	309

Appendices	310
A Setup & Usage	310
A.1 Setting up the environment	310
A.1.1 Installing Anaconda	310
A.1.2 Creating the virtual environment	311
A.1.3 Activating the virtual environment	311
A.2 Recreating the handbook	312
A.2.1 Rendering the book locally:	312
A.2.2 Publishing book to github pages	313
B Working with files and directories	314

Welcome

This is the handbook for [IM939: Data Science Across Disciplines](#), taught at the [Centre for Interdisciplinary Methodologies](#) at the University of Warwick.

Teaching Staff¹:

- Cagatay Turkay (Module leader)
 - Kavin Narasimhan
 - Carlos Cámara-Menoyo

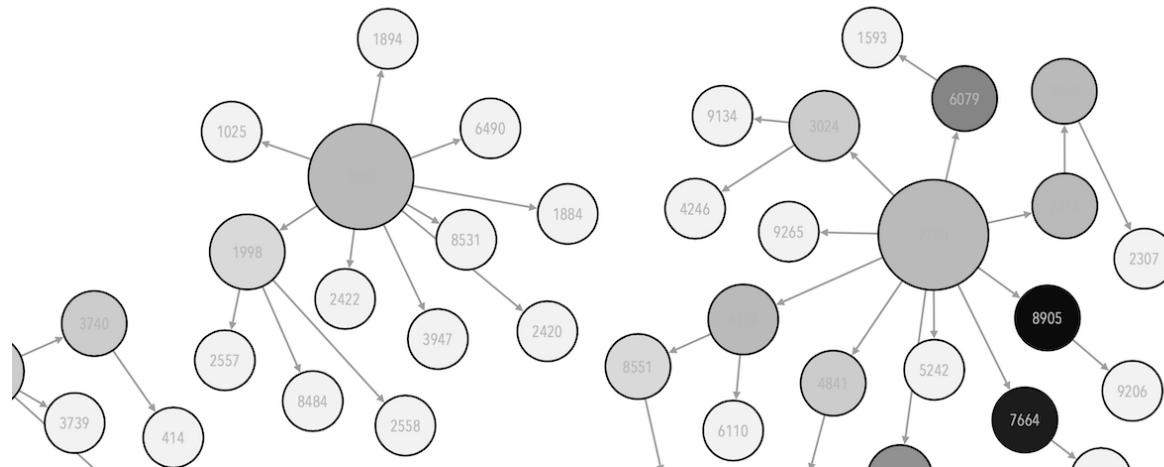


Figure 1: IM939 Logo

What this module is about?

This module introduces students to the fundamental techniques, concepts and contemporary discussions across the broad field of data science. With data and data related artefacts becoming ubiquitous in all aspects of social life, data science gains access to new sources of data, is taken up across an expanding range of research fields and disciplines, and increasingly engages with societal challenges. The module provides an advanced introduction to the theoretical and

¹For a comprehensive list of past and current staff members, refer to [?@sec-staff-past](#)

scientific frameworks of data science, and to the fundamental techniques for working with data using appropriate procedures, algorithms and visualisation. Students learn how to critically approach data and data-driven artefacts, and engage with and critically reflect on contemporary discussions around the practice of data science, its compatibility with different analytics frameworks and disciplinary, and its relation to on-going digital transformations of society. As well as lectures discussing the theoretical, scientific and ethical frameworks of data science, the module features coding labs and workshops that expose students to the practice of working effectively with data, algorithms, and analytical techniques, as well as providing a platform for reflective and critical discussions on data science practices, resulting data artefacts and how they can be interpreted, actioned and influence society.

Course contents

Introduction and historical perspectives (Chapter 1)

This week discusses data science as a field that cuts across disciplines and provides a historical perspective on the subject. We discuss the terms Data Science and Data Scientists, reflect on examples of Data Science projects, and discuss the research process at a methodological level.

Thinking Data: Theoretical and Practical Concerns

This week explores the cultural, ethical, and critical challenges posed by data artefacts and data-intensive scientific processes. Engaging with Critical Data Studies, we discuss issues around data capture, curation, data quality, inclusion/exclusion and representativeness. The session also discusses the different kinds of data that one can encounter across disciplines, the underlying characteristics of data and how we can analytically and practically approach data quality issues and the challenge of identifying and curating appropriate data sets.

...

Acknowledgements

This handbook has been created by Carlos Cámará-Menoyo and Cagatay Turkay, based on the materials from previous years created by Cagatay Turkay and James Tripp.

Licence

Part I

About

Teaching Staff

Cagatay Turkay

Professor, Module Convener

My research falls under the broad area that can be referred to as Visual Data Science and focuses on designing visualisations, interactions and computational methods to enable an effective combination of human and machine capabilities to facilitate data-intensive problem solving.

I have a special interest in working on problems where high-dimensional, spatio-temporal, heterogenous and large datasets are used in answering questions with data.

Kavin Narasimhan

Assistant Professor

Carlos Cámarra-Menoyo

Senior Research Software Engineer

I am a versatile, transdisciplinary, and passionate person with a mixed technical and sociological background. As a Senior Research Software Engineer, I am responsible to support research and teaching by using and developing research software while promoting an open, reproducible research culture and outputs.

Former staff

This module has also been taught by the following people in the past (in alphabetical order):

- **Cagatay Turkay**, Professor (202X-Present)
- **Carlos Cámarra-Menoyo**, Senior Research Software Engineer (2022-Present)
- **James Tripp**, Senior Research Software Engineer (2020-2022)
- **Yulu Pi**, Teaching Assistant (2022-2023)
- **Zofia Bednarowska-Michaiel**, Teaching Fellow (2021-2022)

We would like to thank them all for their contributions to the module, some of which are also reflected in these materials.

Part II

Introduction and Historical Perspectives

1 Session 1



This page is still a stub

This is a placeholder for this content: <https://cagatayturkay.github.io/data-science-across-disciplines/Sessions/session-01.html>

2 IM939 - Lab 1 - Part 1

Welcome to the first Python lab.

Our aim is to introduce a little bit of Python and help you be comfortable with using Jupyter notebooks. No previous knowledge is assumed and questions are encouraged!

Please make sure you have Anaconda Python installed and running on your computer. Instructions for doing this on a Windows 10 machine can be found in the video below. There is a little code before the video which is needed to display the video in the notebook, you can ignore this for now.

For your convenience, the installation files for anaconda for each platform are linked below.

[Windows](#) [MacOS](#)

2.1 Python?

Python is a very popular general purpose programming language. Data scientists use it to clean up, analyse and visualise data. A major strength of python is that the core Python functionality can be extended using libraries. In future labs, you will learn about popular data science libraries such as pandas and numpy.

It is useful to think of programming languages as a structured way to tell the computer what to do. We cover some of the basic features of Python in this lab.

2.1.1 Anaconda

[Anaconda](#) is a collection of different programs. These programs include Python, many of the most popular data science libraries, Jupyter notebooks and development environments such as [VS Code](#) or [Spyder](#), which are Integrated Development Environments (IDEs) that we can use for python.

2.1.2 Jupyter Notebooks

Jupyter notebooks, such as this one, allow you to combine text and code into documents you can edit in the browser. The power of these notebooks is in documenting or describing what you are doing with the code alongside that code. For example, you could detail why you chose a particular clustering algorithm above the clustering code itself. In other words, it adds narrative and helps clarify your workflow.

2.2 Getting started

If you send Python a number then it will print that number for you.

45

45

You will see both the input and output displayed. The input will have a label next to it like 'In [1]' where the number tells you how much code has already been sent to the Python interpreter (the programming interpreting the Python code and returnign the result). A line such as 'In [100]' tells you that 99 code cells have been passed to the Python interpreter in the current session.

Python can carry out simple arithmetic.

44 + 87

131

188 / 12

15.666666666666666

46 - 128

-82

Each time the code in the cell is run and the result from the Python interpreter is displayed.

3 Data Types

3.1 int, floats, strings

Integers are whole numbers. We used them above.

You can also have floats (numbers with decimal points)

```
33.4
```

33.4

and a series of characters (strings).

```
'I have a plan, sir.'
```

```
'I have a plan, sir.'
```

Data types are great and operators such as * do different things depending on the data type. For instance,

```
33 * 3
```

99

That seems quite sensible. What about if we had a string? Run the below line. What is the * doing?

```
'I have a plan, sir' * 3
```

```
'I have a plan, sirI have a plan, sirI have a plan, sir'
```

There are also operators which only work with particular data types.

```
'I have a plan, sir.' / 2
```

```
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

This error message is very informative indeed. It tells us the line which caused the problem and that we have an error. Specifically, our error is a `TypeError`.

The line

'I have a cunning plan' / 2

consists of

string / int

We are trying to divide a string and int. The `/` operand is not able to divide a string by an int.

3.1.1 lists and dictionaries

You can collect multiple values in a list.

```
[35, 'brown', 'yes']
```

```
[35, 'brown', 'yes']
```

Or add keys to the values as a dictionary.

```
{'age':35, 'hair colour': 'brown', 'Glasses': 'yes'}
```

```
{'age': 35, 'hair colour': 'brown', 'Glasses': 'yes'}
```

4 Variables

Variables are bins for storing things in. These things can be data types. For example, the below creates a variable called my_height and stores the in 140 there.

```
my_height = 140
```

The Python interpreter is now storing the int 140 in a bin called my_height. If you pass the variable name to the interpreter then it will behave just like if you typed in 140 instead.

```
my_height
```

```
140
```

```
140
```

Variables are neat when it comes to lists.

```
my_heights = [231, 234, 43]  
my_heights
```

```
[231, 234, 43]
```

```
my_heights[1]
```

```
234
```

Wait, what happened above? What do you think the [1] does?

You can index multiple values from a list.

```
my_heights[0:2]
```

```
[231, 234]
```

4.1 Bringing it all together

What does the below do?

```
radius = 40
pi = 3.1415
circle_area = pi * (radius * radius)

length = 12
square_area = length * length

my_areas = [circle_area, square_area]

my_areas
```

```
[5026.400000000001, 144]
```

As an aside, you can include comments which are not evaluated by the Python interpreter.

```
# this is a number
# another comment
n = 33
```

5 Congratulations

You've reached the end of the first notebook. We've looked at basic data type and variables. These are key components of all programming languages and a key part of working with data.

In the next notebook we will examine using libraries, loading in data, loops and functions.

If you have any questions then you're very welcome to email me at james.tripps@warwick.ac.uk

6 IM939 - Lab 1 - Part 2

Welcome to the second part of IM939 lab 1.

Here we are going to look at libraries, loading in data from a csv file, loops and functions.

6.1 Functions

How do we get the length of a list?

```
lunch_ingredients = ['gravy', 'potatoes', 'meat', 'veg']  
len(lunch_ingredients)
```

4

We used a function. The syntax for a function is `function_name(argument1, argument2)`. In our above example we passed the list `lunch_ingredients` as an argument to the `len` function.

```
james = {'age':35, 'hair colour': 'brown', 'Glasses': 'yes'}  
  
len(james)
```

3

One very useful function is to check a data type. In base R it may be obvious but when you move to libraries, with different data types, it may be worth checking.

```
type(james)  
  
dict  
  
type(lunch_ingredients)
```

```
list  
type(55)
```

```
int
```

6.2 Methods

What is really happening? Well all data types in Python have functions built in. A variable is an object (a string, a list, an int, etc.). Each object has a set of methods do stuff with that object.

For example, I can add a value to the lunch_ingredients list like so

```
lunch_ingredients.append('yorkshire pudding')  
lunch_ingredients  
  
['gravy', 'potatoes', 'meat', 'veg', 'yorkshire pudding']
```

Note the objectname.method notation. By adding a yorkshire pudding, a fine addition to any meal, my lunch ingredients now number 5.

I can count these easily.

```
len(lunch_ingredients)  
  
5
```

Which is the same as the following.

```
lunch_ingredients.__len__()  
  
5
```

You can tab complete to see what methods are available for a given object. There are quite a few [built in python functions](#).

However, do be aware that different objects will likely have different methods. For instance, can we get a length of an int?

```
len(4)
```

```
TypeError: object of type 'int' has no len()
```

The error “`TypeError: object of type ‘int’ has no len()`” essentially means an `int` object has no **len** [dunder method](#).

To see the methods for a given object, type in the object name in a code cell, add a period ‘.’ and press tab.

7 Flow control

7.1 For loops

You may need to repeat some code (such as reading lines in a csv file). For loops allow you to repeat code but with a variable changing each loop.

For instance,

```
for ingredient in lunch_ingredients:  
    print(ingredient)
```

```
gravy  
potatoes  
meat  
veg  
yorkshire pudding
```

The above loop went through each ingredient and printed it.

Note the syntax and spacing. The for statement finishes with a : and the next line has a tab. For loops have to be defined like that. If you miss out the tab then you will be shown an error. It does make the code more readable though.

```
for ingredient in lunch_ingredients:  
    print(ingredient)
```

```
gravy  
potatoes  
meat  
veg  
yorkshire pudding
```

At least the error is informative.

We can have more than one line in the loop.

```
for ingredient in lunch_ingredients:  
    print('I love to eat ' + ingredient)  
    print('food is great')
```

```
I love to eat gravy  
food is great  
I love to eat potatoes  
food is great  
I love to eat meat  
food is great  
I love to eat veg  
food is great  
I love to eat yorkshire pudding  
food is great
```

7.2 If statements

An if statement allows us to run different code depending on a condition.

```
my_num = 3  
  
if my_num > 5:  
    print('Number is bigger than 5')  
  
if my_num < 5:  
    print('Number is less than 5')
```

```
Number is less than 5
```

We can include this in a for loop.

```
for i in range(10): #go through each number 0:10  
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9

for i in range(10):
    print(i)
    if i == 5:
        print('We found a 5!')
```

```
0
1
2
3
4
5
We found a 5!
6
7
8
9
```

You can also include an else part of your if statement.

```
for i in range(10):
    print(i)
    if i == 5:
        print('We found a 5! :)')
    else:
        print('Not a 5. Boo hoo :(')
```

```
0
Not a 5. Boo hoo :(
1
```

```
Not a 5. Boo hoo :(  
2  
Not a 5. Boo hoo :(  
3  
Not a 5. Boo hoo :(  
4  
Not a 5. Boo hoo :(  
5  
We found a 5! :)  
6  
Not a 5. Boo hoo :(  
7  
Not a 5. Boo hoo :(  
8  
Not a 5. Boo hoo :(  
9  
Not a 5. Boo hoo :(
```

8 Modules

A module is a collection of code someone else has written(objects, methods, etc.). A library is a group of modules. In data science work you will use libraries such as numpy and scikit-learn. Anaconda has many of these libraries built in.

Sometimes there are multiple ways to do something. Different libraries may have modules which have similiar functionality but slight differences.

For example, lets go through a comma seperated value file to make some data accessible in Python.

8.1 CSV module

```
import csv
```

The import keyword loads in a module. Here it loads in the csv module from the inbuilt Python library (see the [Python docs on the csv module](#)).

Please put the Facebook.csv file in the same folder as this notebook for this bit.

The code for going through a csv file is a bit lengthy.

```
with open('data/Facebook.csv', mode = 'r', encoding = 'UTF-8') as csvfile:      # open up ou
    reader = csv.reader(csvfile)                                              # create a reader
    for row in reader:                                                        # for each row th
        print(row)                                                          # print out the c
```

```
['\ufe0fftype', 'post_message', 'link', 'link_domain', 'post_published', 'likes_count_fb', 'co
['status', 'Don t forget to post your photographs from heritage weekend of you having a good
['status', 'If you want to learn about Historic Coventry this Saturday we will be carrying on
['event', '3 more dates for September Tours of The Medieval City Wall Below. Saturday 16th Sept
['link', 'http://www.coventrytelegraph.net/whats-on/arts-culture-news/majestic-medieval-hall-1
['status', 'Different angle of St Michaels', '', '', '2017-08-26T09:04:35+0000', '11', '0',
['link', 'The Heritage Open Days are taking place from 7-10 September with venues opening th
['status', 'Hi thanks to be able to share here & be part of tnis group ; Good to kbow iof yt
```

```

['link', 'http://www.coventrytelegraph.net/whats-on/whats-on-news/future-coventrys-albany-the-new-look-for-the-citys-new-arts-and-culture-centre-1.103444221'],
['photo', 'https://www.facebook.com/peter.garbett.9/posts/1769150796433812 Sad we are loosing'],
['link', 'https://www.facebook.com/CoventryPVC/posts/859986554155927', 'http://www.crowdfund.co.uk'],
['photo', 'Looking good in the sunshine.. https://www.facebook.com/visitcoventry.net/posts/5023444444444444'],
['status', 'Best unique Pubs Whats your favourite pubs and why? in Coventry and surrounding areas'],
['status', 'Thank you for letting me join your group', '', '', '2017-07-19T14:30:58+0000', ''],
['link', 'https://www.facebook.com/theprideofwillenhall/posts/1378187435629358', 'http://www.visitcoventry.net'],
['link', 'https://www.facebook.com/peter.garbett.9/posts/1740635729285319 Coventry shortlist'],
['photo', 'Nice. https://www.facebook.com/visitcoventry.net/photos/a.363722627310847.107374186'],
['photo', 'I ve always wanted to have a nose down this passageway.. https://www.facebook.com/visitcoventry.net'],
['link', 'http://www.coventrytelegraph.net/whats-on/music-nightlife-news/what-godiva-festival'],
['video', 'Looks like a great idea.', 'https://www.facebook.com/stmarysguildhall/videos/1015422627310847'],
['photo', 'Lovely pic.. https://www.facebook.com/visitcoventry.net/photos/a.363722627310847.107374186'],
['status', 'What is your favourite restaurant or eatery in Coventry area?', '', '', '2017-06-19T14:30:58+0000'],
['photo', 'Interesting.. https://www.facebook.com/visitcoventry.net/photos/a.363722627310847.107374186'],
['link', 'https://www.facebook.com/BerkswellWindmill/posts/1322847697832828', 'https://www.facebook.com/visitcoventry.net'],
['status', 'Just saw a post about people that made me think would almost be a great motto for the city'],
['event', 'Why not make a weekend of it? It s Jaguar Super Saturday! In aid of Air Ambulance'],
['event', ':) we have a wonderful collection of exciting Citroen motorcars at the Museum on Saturday'],
['event', 'City Wall Guided Costume Tour this Saturday meeting at 2pm at Priory Visitors Centre'],
['link', 'https://www.facebook.com/BerkswellWindmill/posts/1322847697832828', 'https://www.facebook.com/visitcoventry.net'],

```

Ok, each row looks like a list. But printing it out is pretty messy. We should dump this into a list of dictionaries (so we can pick out particular values).

```

csv_content = [] # create list to put our data into

with open('data/Facebook.csv', mode = 'r', encoding = 'UTF-8') as csvfile: # open up csv file
    reader = csv.DictReader(csvfile) # create a reader with the data
    for row in reader: # for each row
        csv_content.append(row)
        print(row) # put the created dictionary in the list

```

```

{'\ufe0fftype': 'status', 'post_message': 'Don t forget to post your photographs from heritage open days!'},
{'\ufe0fftype': 'status', 'post_message': 'If you want to learn about Historic Coventry this Saturday, come along to the Coventry Open Days!'},
{'\ufe0fftype': 'event', 'post_message': '3 more dates for September Tours of The Medieval City Walls!'},
{'\ufe0fftype': 'link', 'post_message': 'http://www.coventrytelegraph.net/whats-on/arts-culture/heritage-open-days-1.103444221'},
{'\ufe0fftype': 'status', 'post_message': 'Different angle of St Michaels', 'link': '', 'link_type': 'Image'},
{'\ufe0fftype': 'link', 'post_message': 'The Heritage Open Days are taking place from 7-10 September!'},
{'\ufe0fftype': 'status', 'post_message': 'Hi thanks to be able to share here & be part of the Coventry Open Days!'},
{'\ufe0fftype': 'link', 'post_message': 'http://www.coventrytelegraph.net/whats-on/whats-on-news/heritage-open-days-1.103444221'},
{'\ufe0fftype': 'photo', 'post_message': 'https://www.facebook.com/peter.garbett.9/posts/1769150796433812'}

```

Look at the first entry in csv_content.

```
csv_content[0]
```

```
{'\ufefftype': 'status',
'post_message': 'Don t forget to post your photographs from heritage weekend of you having',
'link': '',
'link_domain': '',
'post_published': '2017-09-09T13:57:14+0000',
'likes_count_fb': '5',
'comments_count_fb': '1',
'reactions_count_fb': '5',
'shares_count_fb': '0',
'engagement_fb': '6'}
```

Dictionaries have keys. The `keys` method of the dictionary object will show them to us (though they are obvious from printing the first line above).

```
csv_content[0].keys()
```

```
dict_keys(['ufefftype', 'post_message', 'link', 'link_domain', 'post_published', 'likes_count'])
```

```
csv_content[0]['post_message']
```

'Don t forget to post your photographs from heritage weekend of you having a good time on he

We have a list where each list element is a dictionary. So we need to index our list each time, hence the csv_content[0].

To go through our list we need to use a for loop. So, in order to get each post message.

```
for post in csv_content:  
    print(post['post_message'])
```

Don t forget to post your photographs from heritage weekend of you having a good time on he
If you want to learn about Historic Coventry this Saturday we will be carrying out a guided 1
3 more dates for September Tours of The Medieval City Wall Below. Saturday 16th 23 rd and 30th
<http://www.coventrytelegraph.net/whats-on/arts-culture-news/majestic-medieval-hall-coventrys>
Different angle of St Michaels

The Heritage Open Days are taking place from 7-10 September with venues opening their doors
Hi thanks to be able to share here & be part of tnis group ; Good to kbow iof yt area as goin
<http://www.coventrytelegraph.net/whats-on/whats-on-news/future-coventrys-albany-theatre-been>
<https://www.facebook.com/peter.garbett.9/posts/1769150796433812> Sad we are loosing this mus
<https://www.facebook.com/CoventryPVC/posts/859986554155927>

Looking good in the sunshine.. <https://www.facebook.com/visitcoventry.net/posts/501006916915>
Best unique Pubs Whats your favourite pubs and why? in Coventry and surrounding areas.

Thank you for letting me join your group

<https://www.facebook.com/theprideofwillenhall/posts/1378187435629358>

<https://www.facebook.com/peter.garbett.9/posts/1740635729285319> Coventry shortlisted!

Nice. <https://www.facebook.com/visitcoventry.net/photos/a.363722627310847.1073741828.3619296>

I ve always wanted to have a nose down this passageway.. <https://www.facebook.com/visitcoventry.net/photos/a.363722627310847.1073741828.3619296>

<http://www.coventrytelegraph.net/whats-on/music-nightlife-news/what-godiva-festival-weather->

Looks like a great idea.

Lovely pic.. <https://www.facebook.com/visitcoventry.net/photos/a.363722627310847.1073741828.3619296>

What is your favourite restaurant or eatery in Coventry area?

Interesting.. <https://www.facebook.com/visitcoventry.net/photos/a.363722627310847.1073741828.3619296>

<https://www.facebook.com/BerkswellWindmill/posts/1322847697832828>

Just saw a post about people that made me think would almost be a great motto for our fabulou

Why not make a weekend of it? It s Jaguar Super Saturday! In aid of Air Ambulance details on

:) we have a wonderful collection of exciting Citroen motorcars at the Museum on Sunday if you

City Wall Guided Costume Tour this Saturday meeting at 2pm at Priory Visitors Centre (behind

<https://www.facebook.com/BerkswellWindmill/posts/1322847697832828>

If I want to do data science, where accessing data in a sensible way is key, then there must be a better way! There is.

9 Pandas

The Pandas library is designed with data science in mind. You will examine it more in the coming weeks. Reading CSV files with pandas is very easy.

```
import pandas as pd
```

The import x as y is good practice. In the below code, anything with a pd. prefix comes from pandas. This is particularly useful for preventing a module from overwriting inbuilt Python functionality.

```
df = pd.read_csv('data/Facebook.csv', encoding = 'UTF-8')
```

That was easy. We can even pick out specific rows.

```
df['post_message']
```

```
0    Don t forget to post your photographs from her...
1    If you want to learn about Historic Coventry t...
2    3 more dates for September Tours of The Mediev...
3    http://www.coventrytelegraph.net/whats-on/arts...
4                      Different angle of St Michaels
5    The Heritage Open Days are taking place from 7...
6    Hi thanks to be able to share here & be part o...
7    http://www.coventrytelegraph.net/whats-on/what...
8    https://www.facebook.com/peter.garbett.9/posts...
9    https://www.facebook.com/CoventryPVC/posts/859...
10   Looking good in the sunshine.. https://www.fac...
11   Best unique Pubs Whats your favourite pubs and...
12   Thank you for letting me join your group
13   https://www.facebook.com/theprideofwillenhall/...
14   https://www.facebook.com/peter.garbett.9/posts...
15   Nice. https://www.facebook.com/visitcoventry.n...
16   I ve always wanted to have a nose down this pa...
17   http://www.coventrytelegraph.net/whats-on/musi...
18   Looks like a great idea.
```

```

19  Lovely pic.. https://www.facebook.com/visitcov...
20  What is your favourite restaurant or eatery in...
21  Interesting.. https://www.facebook.com/visitco...
22  https://www.facebook.com/BerkswellWindmill/pos...
23  Just saw a post about people that made me thin...
24  Why not make a weekend of it? It's Jaguar Supe...
25  :) we have a wonderful collection of exciting ...
26  City Wall Guided Costume Tour this Saturday me...
27  https://www.facebook.com/BerkswellWindmill/pos...
Name: post_message, dtype: object

```

And even look at the dataset in a pretty way.

```
df
```

	type	post_message	link
0	status	Don't forget to post your photographs from her...	NaN
1	status	If you want to learn about Historic Coventry t...	NaN
2	event	3 more dates for September Tours of The Medieval...	https://www.facebook.com/events/1935...
3	link	http://www.coventrytelegraph.net/whats-on/arts...	http://www.coventrytelegraph.net/what...
4	status	Different angle of St Michaels	NaN
5	link	The Heritage Open Days are taking place from 7...	http://www.coventry.gov.uk/hod
6	status	Hi thanks to be able to share here & be part o...	NaN
7	link	http://www.coventrytelegraph.net/whats-on/what...	http://www.coventrytelegraph.net/what...
8	photo	https://www.facebook.com/peter.garbett.9/posts...	https://www.facebook.com/photo.php?...
9	link	https://www.facebook.com/CoventryPVC/posts/859...	http://www.crowdfunder.co.uk/coventry...
10	photo	Looking good in the sunshine.. https://www.fac...	https://www.facebook.com/visitcoventry...
11	status	Best unique Pubs Whats your favourite pubs and...	NaN
12	status	Thank you for letting me join your group	NaN
13	link	https://www.facebook.com/theprideofwillenhall/...	http://www.coventrytelegraph.net/what...
14	link	https://www.facebook.com/peter.garbett.9/posts...	https://coventry2021.co.uk/news/?is_m...
15	photo	Nice. https://www.facebook.com/visitcoventry.n...	https://www.facebook.com/visitcoventry...
16	photo	I've always wanted to have a nose down this pa...	https://www.facebook.com/visitcoventry...
17	link	http://www.coventrytelegraph.net/whats-on/musi...	http://www.coventrytelegraph.net/what...
18	video	Looks like a great idea.	https://www.facebook.com/stmarysguild...
19	photo	Lovely pic.. https://www.facebook.com/visitcov...	https://www.facebook.com/visitcoventry...
20	status	What is your favourite restaurant or eatery in...	NaN
21	photo	Interesting.. https://www.facebook.com/visitco...	https://www.facebook.com/visitcoventry...
22	link	https://www.facebook.com/BerkswellWindmill/pos...	https://www.facebook.com/BerkswellW...
23	status	Just saw a post about people that made me thin...	NaN
24	event	Why not make a weekend of it? It's Jaguar Supe...	https://www.facebook.com/events/2013...

type	post_message	link
25	event	:) we have a wonderful collection of exciting ...
26	event	City Wall Guided Costume Tour this Saturday me...
27	link	https://www.facebook.com/BerkswellWindmill/post/3620431352043135

Huzzah!

A final point, pd.read_csv returns a pandas specific object with associated methods.

```
type(df)
```

```
pandas.core.frame.DataFrame
```

```
df.dtypes
```

```
type          object
post_message  object
link          object
link_domain   object
post_published object
likes_count_fb int64
comments_count_fb int64
reactions_count_fb int64
shares_count_fb int64
engagement_fb  int64
dtype: object
```

Where int64 are integers and object here refers to strings.

Part III

Thinking Data: Theoretical and Practical Concerns

10 IM939 Lab 2 - Part 1

Last session we loaded data using [Pandas](#). Here we explore how to use Pandas to read in, process and explore data.

As before we load pandas and use the `read_csv` method.

```
import pandas as pd

df = pd.read_csv('data/raw/office_ratings.csv', encoding='UTF-8')

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 188 entries, 0 to 187
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
---  --  
 0   season      188 non-null    int64  
 1   episode     188 non-null    int64  
 2   title       188 non-null    object  
 3   imdb_rating 188 non-null    float64 
 4   total_votes 188 non-null    int64  
 5   air_date    188 non-null    object  
dtypes: float64(1), int64(3), object(2)
memory usage: 8.9+ KB
```

10.1 Help!

Python has inbuilt documentation. To access this add a `?` before an object or method.

For example, our dataframe

```
?df.info
```

The dtypes property (properties of object are values associated with the object and are not called with a () at the end).

```
?df.dtypes
```

The info method for dataframes.

```
?df.info
```

The below is quite long. But goes give you the various arguments (options) you can use with the method.

```
?pd.read_csv
```

The Pandas documentation is rather good. Relevant to our below work is:

- [What kind of data does pandas handle?](#)
- [How to calculate summary statistics?](#)
- [How to create plots in pandas?](#)
- [How to handle time series data with ease?](#)

I also found [a rather nice series of lessons a kind person put together](#). There are lots of online tutorials which will help you.

10.2 Structure

How do we find out the structure of our data?

Well, the variable df is now a pandas DataFrame object.

```
type(df)
```

```
pandas.core.frame.DataFrame
```

The DataFrame object has lots of built in methods and attributes.

The info method gives us information about datatypes, dimensions and the presence of null values in our dataframe.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 188 entries, 0 to 187
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
---  --  
 0   season      188 non-null    int64  
 1   episode     188 non-null    int64  
 2   title       188 non-null    object  
 3   imdb_rating 188 non-null    float64 
 4   total_votes 188 non-null    int64  
 5   air_date    188 non-null    object  
dtypes: float64(1), int64(3), object(2)
memory usage: 8.9+ KB
```

We can just look at the datatypes if we want.

```
df.dtypes
```

```
season      int64
episode     int64
title       object
imdb_rating float64
total_votes int64
air_date    object
dtype: object
```

Or just the dimensions.

```
df.shape
```

```
(188, 6)
```

In this case, there are only 188 rows. But for larger datasets we might want to look at the head (top 5) and tail (bottom 5) rows.

```
df.head()
```

	season	episode	title	imdb_rating	total_votes	air_date
0	1	1	Pilot	7.6	3706	2005-03-24
1	1	2	Diversity Day	8.3	3566	2005-03-29
2	1	3	Health Care	7.9	2983	2005-04-05
3	1	4	The Alliance	8.1	2886	2005-04-12
4	1	5	Basketball	8.4	3179	2005-04-19

```
df.tail()
```

	season	episode	title	imdb_rating	total_votes	air_date
183	9	19	Stairmageddon	8.0	1484	2013-04-11
184	9	20	Paper Airplane	8.0	1482	2013-04-25
185	9	21	Livin' the Dream	8.9	2041	2013-05-02
186	9	22	A.A.R.M.	9.3	2860	2013-05-09
187	9	23	Finale	9.7	7934	2013-05-16

We may need to put together a dataset.

Consider the two dataframe below

```
df_1 = pd.read_csv('data/raw/office1.csv', encoding='UTF-8')
df_2 = pd.read_csv('data/raw/office2.csv', encoding='UTF-8')
```

```
df_1.head()
```

	id	season	episode	imdb_rating
0	5-1	5	1	8.8
1	9-13	9	13	7.7
2	5-6	5	6	8.5
3	3-23	3	23	9.3
4	9-16	9	16	8.2

```
df_2.head()
```

	id	total_votes
0	4-10	2095

	id	total_votes
1	3-21	2403
2	7-24	2040
3	6-18	1769
4	8-8	1584

The total votes and imdb ratings data are split between files. There is a common column called id. We can join the two dataframes together using the common column.

```
inner_join_office_df = pd.merge(df_1, df_2, on='id', how='inner')
inner_join_office_df
```

	id	season	episode	imdb_rating	total_votes
0	5-1	5	1	8.8	2501
1	9-13	9	13	7.7	1394
2	5-6	5	6	8.5	2018
3	3-23	3	23	9.3	3010
4	9-16	9	16	8.2	1572
...
183	5-21	5	21	8.7	2032
184	2-13	2	13	8.3	2363
185	9-6	9	6	7.8	1455
186	2-2	2	2	8.2	2736
187	3-4	3	4	8.0	2311

In this way you can combine datasets using common columns. We will leave that for the moment. If you want more information about merging data then see [this page](#) and the [pandas documentation](#).

10.3 Summary

To get an overview of our data we can ask Python to ‘describe our data’

```
df.describe()
```

	season	episode	imdb_rating	total_votes
count	188.000000	188.000000	188.000000	188.000000
mean	5.468085	11.877660	8.257447	2126.648936
std	2.386245	7.024855	0.538067	787.098275
min	1.000000	1.000000	6.700000	1393.000000
25%	3.000000	6.000000	7.900000	1631.500000
50%	6.000000	11.500000	8.200000	1952.500000
75%	7.250000	18.000000	8.600000	2379.000000
max	9.000000	26.000000	9.700000	7934.000000

or we can pull out specific statistics.

```
df.mean(numeric_only=True)
```

```
season      5.468085
episode     11.877660
imdb_rating 8.257447
total_votes 2126.648936
dtype: float64
```

Note the error triggered above due to pandas attempting to calculate the mean of the wrong type (i.e. non-numeric values). We can address that by only computing the mean of numeric values (see below):

```
df.mean(numeric_only=True)
```

```
season      5.468085
episode     11.877660
imdb_rating 8.257447
total_votes 2126.648936
dtype: float64
```

or the sum.

```
df.sum()
```

```
season          1028
episode         2233
```

```
title          PilotDiversity DayHealth CareThe AllianceBaske...
imdb_rating          1552.4
total_votes          399810
air_date      2005-03-242005-03-292005-04-052005-04-122005-0...
dtype: object
```

Similarly to what happened with `mean()`, `sum()` is adding all values in every observation of every attribute, regardless of their type. Can you see what happens with strings? And with dates?

Again, we can only force to use numeric values only:

```
df.sum(numeric_only=True)
```

```
season          1028.0
episode          2233.0
imdb_rating      1552.4
total_votes      399810.0
dtype: float64
```

Calculating these statistics for specific columns is straight forward.

```
df['imdb_rating'].mean()
```

```
8.25744680851064
```

```
df['total_votes'].sum()
```

```
399810
```

10.4 Selecting subsets

We can even select more than one column!

```
df[['imdb_rating', 'total_votes']].mean()
```

```
imdb_rating      8.257447
total_votes     2126.648936
dtype: float64
```

Two sets of squared brackets is needed because you are passing a list of the column names to the getitem dunder method of the pandas dataframe object (thank [this stackoverflow question](#)). You can also check out the pandas documentation on [indexing and selecting data](#).

You can also select by row and column name using the iloc method. You can specify the [row, column]. So to choose the value in the 2nd row and 4th column.

```
df.iloc[2,4]
```

```
2983
```

All the rows or all the columns are indicated by `[:,]`. Such as,

```
df.iloc[:,2]
```

```
0             Pilot
1      Diversity Day
2      Health Care
3      The Alliance
4      Basketball
...
183     Stairmageddon
184     Paper Airplane
185  Livin' the Dream
186      A.A.R.M.
187      Finale
Name: title, Length: 188, dtype: object
```

```
df.iloc[2,:]
```

```
season          1
episode         3
title      Health Care
imdb_rating     7.9
total_votes     2983
air_date  2005-04-05
Name: 2, dtype: object
```

We can use negative values in indexes to indicate ‘from the end’. So, an index of [-10, :] returns the 10th from last row.

```
df.iloc[-10, :]
```

```
season           9
episode          14
title  Vandalism
imdb_rating     7.6
total_votes     1402
air_date  2013-01-31
Name: 178, dtype: object
```

Instead of using tail, we could ask for the last 10 rows with an index of [-10:, :]. I read : as ‘and everything else’ in these cases.

```
df.iloc[-5:, :]
```

	season	episode	title	imdb_rating	total_votes	air_date
183	9	19	Stairmageddon	8.0	1484	2013-04-11
184	9	20	Paper Airplane	8.0	1482	2013-04-25
185	9	21	Livin' the Dream	8.9	2041	2013-05-02
186	9	22	A.A.R.M.	9.3	2860	2013-05-09
187	9	23	Finale	9.7	7934	2013-05-16

```
df.tail()
```

	season	episode	title	imdb_rating	total_votes	air_date
183	9	19	Stairmageddon	8.0	1484	2013-04-11
184	9	20	Paper Airplane	8.0	1482	2013-04-25
185	9	21	Livin' the Dream	8.9	2041	2013-05-02
186	9	22	A.A.R.M.	9.3	2860	2013-05-09
187	9	23	Finale	9.7	7934	2013-05-16

Note that the row is shown on the left. That will stop you getting lost in slices of the data.

For the top ten rows

```
df.iloc[:10,:]
```

	season	episode	title	imdb_rating	total_votes	air_date
0	1	1	Pilot	7.6	3706	2005-03-24
1	1	2	Diversity Day	8.3	3566	2005-03-29
2	1	3	Health Care	7.9	2983	2005-04-05
3	1	4	The Alliance	8.1	2886	2005-04-12
4	1	5	Basketball	8.4	3179	2005-04-19
5	1	6	Hot Girl	7.8	2852	2005-04-26
6	2	1	The Dundies	8.7	3213	2005-09-20
7	2	2	Sexual Harassment	8.2	2736	2005-09-27
8	2	3	Office Olympics	8.4	2742	2005-10-04
9	2	4	The Fire	8.4	2713	2005-10-11

Of course, we can run methods on these slices. We could, if we wanted to, calculate the mean imdb rating of only the first and last 100 episodes. *Note* the indexing starts at 0 so we want the column index of 3 (0:season, 1:episode, 2:title, 3:imdb_rating).

```
df.iloc[:100,3].mean()
```

8.483

```
df.iloc[-100:,3].mean()
```

8.062

If you are unsure how many rows you have then the count method comes to the rescue.

```
df.iloc[-100:,3].count()
```

100

```
df.describe()
```

	season	episode	imdb_rating	total_votes
count	188.000000	188.000000	188.000000	188.000000
mean	5.468085	11.877660	8.257447	2126.648936
std	2.386245	7.024855	0.538067	787.098275
min	1.000000	1.000000	6.700000	1393.000000
25%	3.000000	6.000000	7.900000	1631.500000
50%	6.000000	11.500000	8.200000	1952.500000
75%	7.250000	18.000000	8.600000	2379.000000
max	9.000000	26.000000	9.700000	7934.000000

So it looks like the last 100 episodes were less good than the first 100. I guess that is why it was cancelled.

Our data is organised by season. Looking at the average by season might help.

```
df[['season', 'imdb_rating']].groupby('season').mean()
```

season	imdb_rating
1	8.016667
2	8.436364
3	8.573913
4	8.600000
5	8.492308
6	8.219231
7	8.316667
8	7.666667
9	7.956522

The above line groups our dataframe by values in the season column and then displays the mean for each group. Pretty nifty.

Season 8 looks pretty bad. We can look at just the rows for season 8.

```
df[df['season'] == 8]
```

	season	episode	title	imdb_rating	total_votes	air_date
141	8	1	The List	8.2	1829	2011-09-22
142	8	2	The Incentive	8.2	1668	2011-09-29

	season	episode	title	imdb_rating	total_votes	air_date
143	8	3	Lotto	7.3	1601	2011-10-06
144	8	4	Garden Party	8.1	1717	2011-10-13
145	8	5	Spooked	7.6	1543	2011-10-27
146	8	6	Doomsday	7.8	1476	2011-11-03
147	8	7	Pam's Replacement	7.7	1563	2011-11-10
148	8	8	Gettysburg	7.0	1584	2011-11-17
149	8	9	Mrs. California	7.7	1553	2011-12-01
150	8	10	Christmas Wishes	8.0	1547	2011-12-08
151	8	11	Trivia	7.9	1488	2012-01-12
152	8	12	Pool Party	8.0	1612	2012-01-19
153	8	13	Jury Duty	7.5	1478	2012-02-02
154	8	14	Special Project	7.8	1432	2012-02-09
155	8	15	Tallahassee	7.9	1522	2012-02-16
156	8	16	After Hours	8.1	1567	2012-02-23
157	8	17	Test the Store	7.8	1478	2012-03-01
158	8	18	Last Day in Florida	7.8	1429	2012-03-08
159	8	19	Get the Girl	6.7	1642	2012-03-15
160	8	20	Welcome Party	7.2	1489	2012-04-12
161	8	21	Angry Andy	7.1	1585	2012-04-19
162	8	22	Fundraiser	7.1	1453	2012-04-26
163	8	23	Turf War	7.7	1393	2012-05-03
164	8	24	Free Family Portrait Studio	7.8	1464	2012-05-10

We can get an overview of the rating of all chapters within season 8 by:

```
df.loc[df['season'] == 8, 'imdb_rating'].describe()
```

```
count    24.000000
mean     7.666667
std      0.405041
min      6.700000
25%     7.450000
50%     7.800000
75%     7.925000
max     8.200000
Name: imdb_rating, dtype: float64
```

```
df['season'] == 8
```

```

0      False
1      False
2      False
3      False
4      False
...
183     False
184     False
185     False
186     False
187     False
Name: season, Length: 188, dtype: bool

```

Generally pretty bad, but there is clearly one very disliked episode.

10.5 Adding columns

We can add new columns pretty simply.

```

df['x'] = 44
df.head()

```

	season	episode	title	imdb_rating	total_votes	air_date	x
0	1	1	Pilot	7.6	3706	2005-03-24	44
1	1	2	Diversity Day	8.3	3566	2005-03-29	44
2	1	3	Health Care	7.9	2983	2005-04-05	44
3	1	4	The Alliance	8.1	2886	2005-04-12	44
4	1	5	Basketball	8.4	3179	2005-04-19	44

Our new column can be an operation on other columns

```

df['rating_div_total_votes'] = df['imdb_rating'] / df['total_votes']
df.head()

```

	season	episode	title	imdb_rating	total_votes	air_date	x	rating_div_total_votes
0	1	1	Pilot	7.6	3706	2005-03-24	44	0.002051
1	1	2	Diversity Day	8.3	3566	2005-03-29	44	0.002328
2	1	3	Health Care	7.9	2983	2005-04-05	44	0.002648

	season	episode	title	imdb_rating	total_votes	air_date	x	rating_div_total_votes
3	1	4	The Alliance	8.1	2886	2005-04-12	44	0.002807
4	1	5	Basketball	8.4	3179	2005-04-19	44	0.002642

or as simple as adding one to every value.

```
df['y'] = df['season'] + 1
df.iloc[0:5,:]
```

	season	episode	title	imdb_rating	total_votes	air_date	x	rating_div_total_votes
0	1	1	Pilot	7.6	3706	2005-03-24	44	0.002051
1	1	2	Diversity Day	8.3	3566	2005-03-29	44	0.002328
2	1	3	Health Care	7.9	2983	2005-04-05	44	0.002648
3	1	4	The Alliance	8.1	2886	2005-04-12	44	0.002807
4	1	5	Basketball	8.4	3179	2005-04-19	44	0.002642

```
y = df['season'] + 1
```

10.6 Writing data

Pandas supports writing out data frames to various formats.

```
?df.to_csv
```

Now you can uncomment the code below to save your dataframe into a csv file. But before doing so, check that your `data/output` folder is empty:

```
# df.to_csv('data/output/my_output_ratings.csv', encoding='UTF-8')
```

```
?df.to_excel
```

Now you can uncomment the code below to save your dataframe into an excel file. But before doing so, check that your `data/output` folder is empty:

```
# df.to_excel('data/output/my_output_ratings.xlsx')
```

11 IM939 Lab 2 - Part 2

It is far easier to look at trends in data by creating plots. Below we do just that and briefly look at plotting data by date.

```
import pandas as pd
df = pd.read_csv('data/raw/office_ratings.csv', encoding='UTF-8')

df.head()
```

	season	episode	title	imdb_rating	total_votes	air_date
0	1	1	Pilot	7.6	3706	2005-03-24
1	1	2	Diversity Day	8.3	3566	2005-03-29
2	1	3	Health Care	7.9	2983	2005-04-05
3	1	4	The Alliance	8.1	2886	2005-04-12
4	1	5	Basketball	8.4	3179	2005-04-19

11.1 Plots

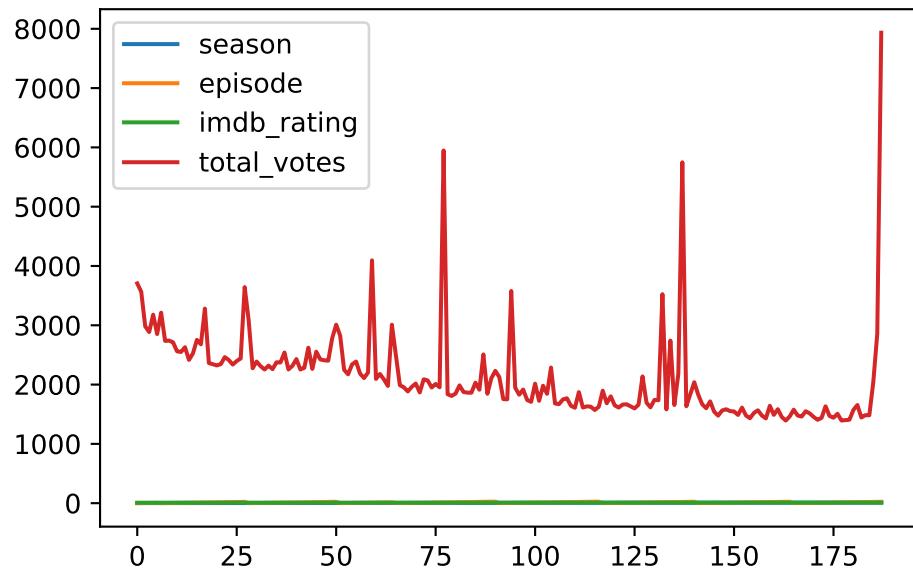
11.2 Univariate - a single variable

Plots are a great way to see trends.

```
?df.plot
```

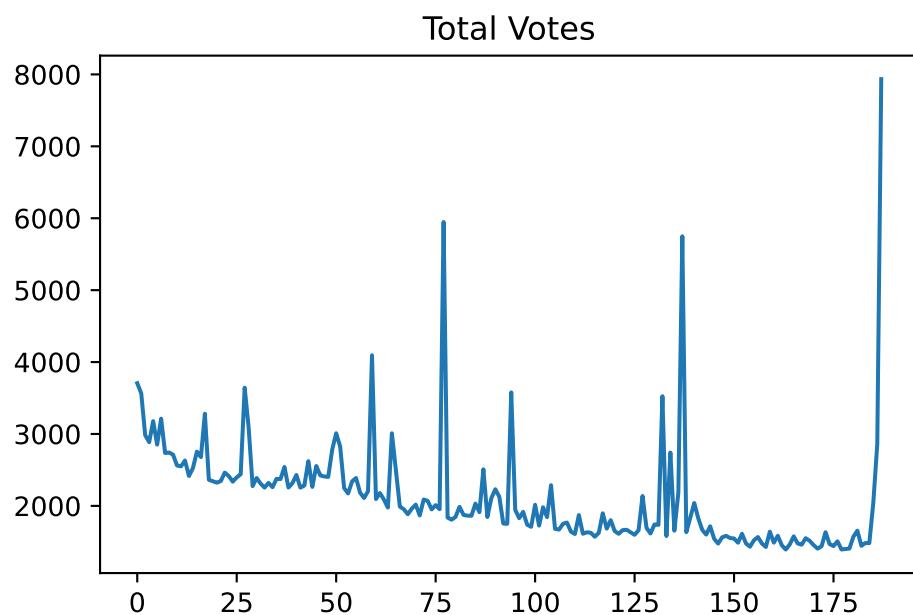
```
df.plot()
```

```
<Axes: >
```



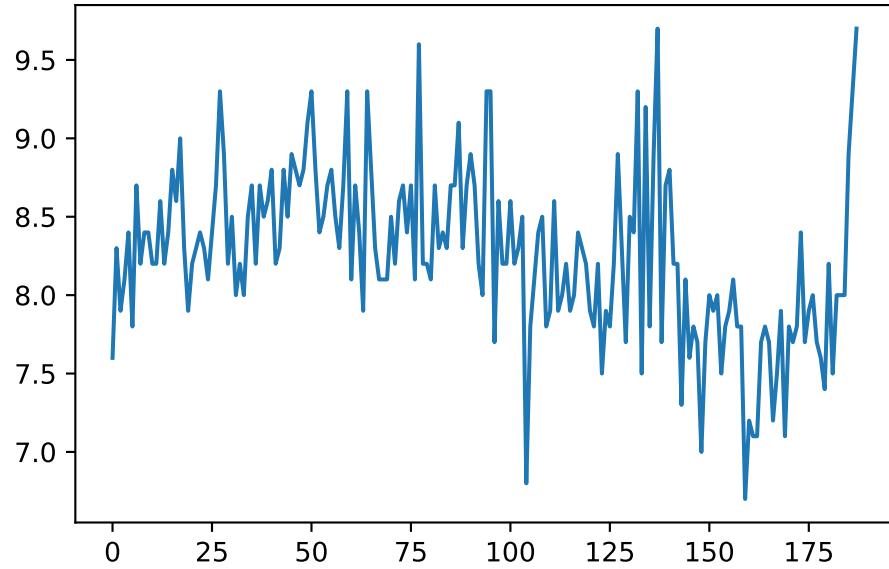
We can look at points instead of lines.

```
df['total_votes'].plot(title='Total Votes')  
<Axes: title={'center': 'Total Votes'}>
```



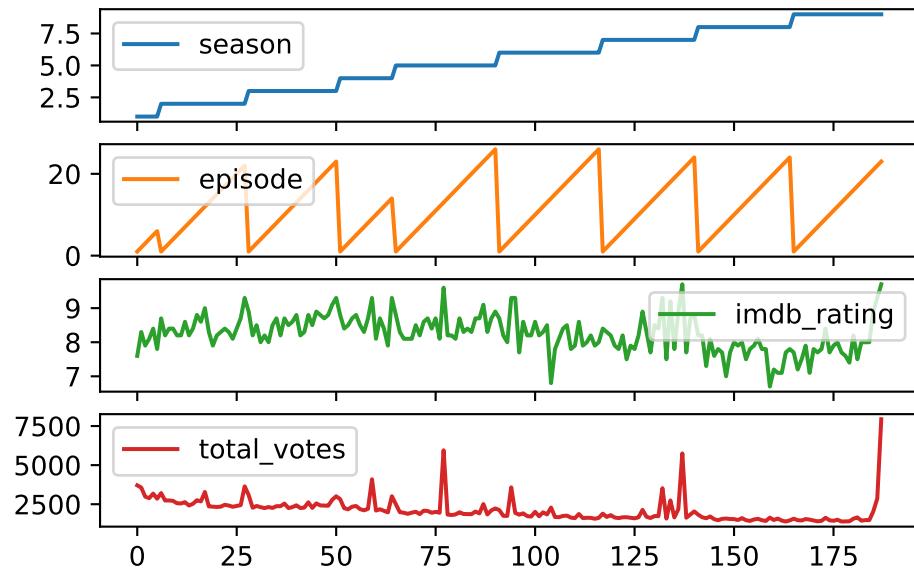
```
?df.plot  
df['imdb_rating'].plot()
```

<Axes: >



Or we could create subplots.

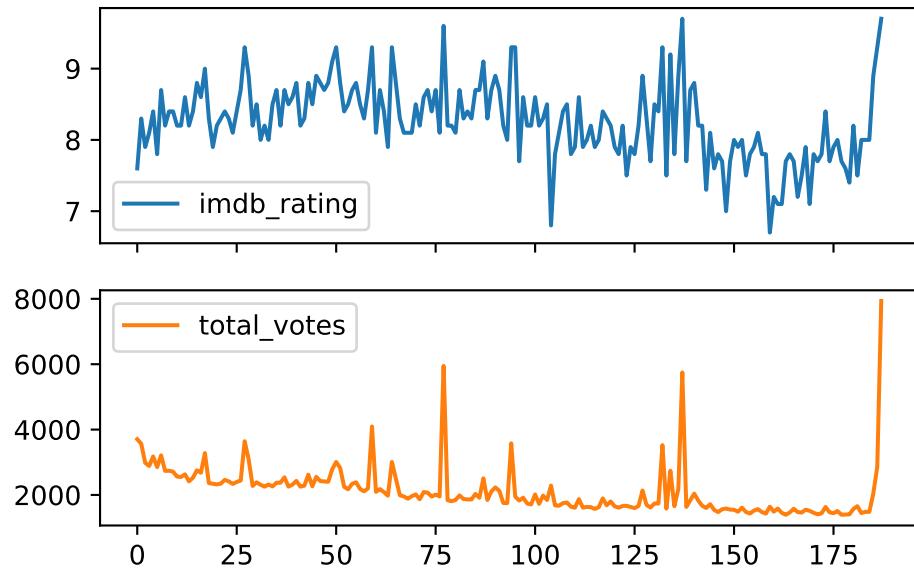
```
df.plot(subplots=True)  
  
array([<Axes: >, <Axes: >, <Axes: >, <Axes: >], dtype=object)
```



Season and episode is not at all informative here.

```
df[['imdb_rating', 'total_votes']].plot(subplots=True)
```

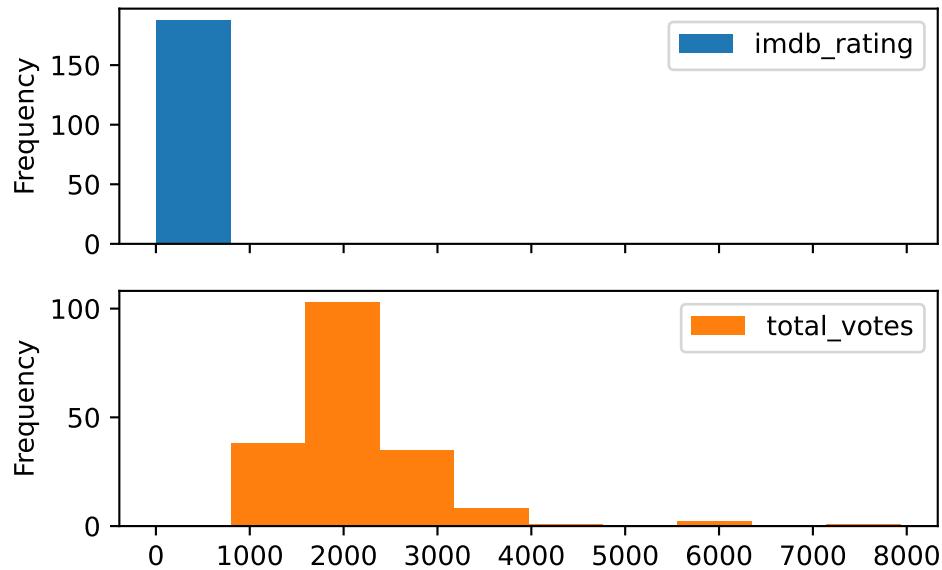
```
array([<Axes: >, <Axes: >], dtype=object)
```



```
?df.plot

df[['imdb_rating', 'total_votes']].plot(subplots=True, kind='hist')

array([<Axes: ylabel='Frequency'>, <Axes: ylabel='Frequency'>],
      dtype=object)
```

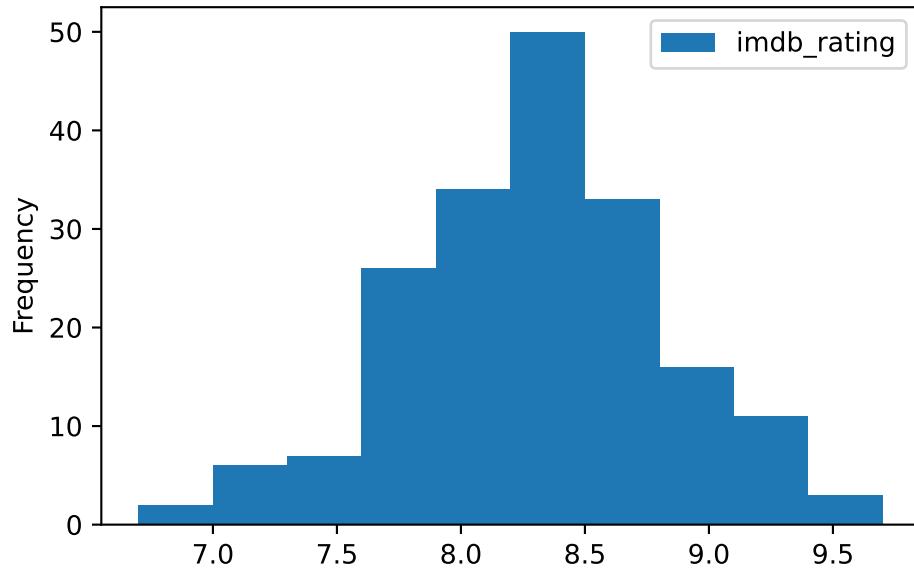


Unfortunatly, our x axis is bunched up. The above tells us that the all our IMDB ratings are between 0 and a little less than 1000... not useful.

Probably best to plot them individually.

```
df[['imdb_rating']].plot(kind='hist')

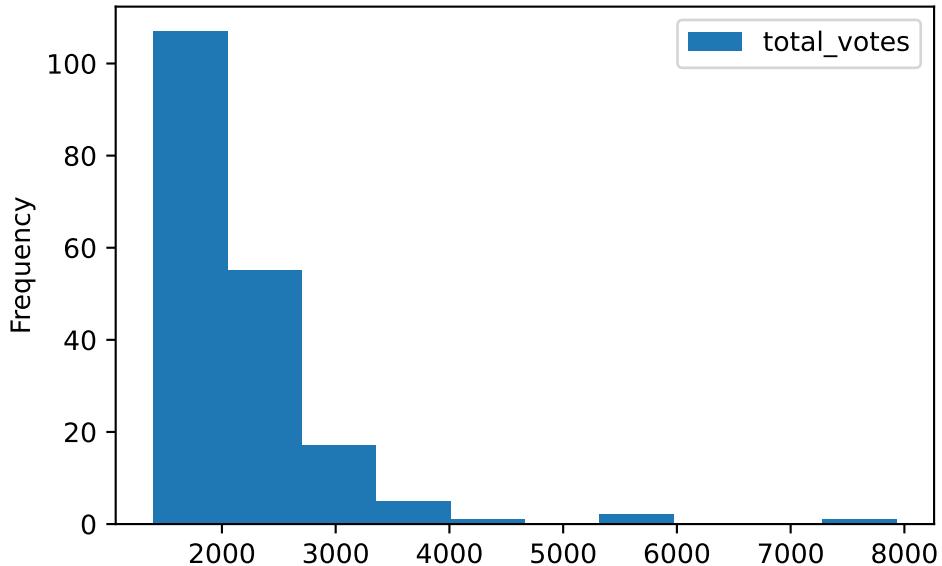
<Axes: ylabel='Frequency'>
```



Quite a sensible gaussian shape (a central point with the frequency decreasing symmetrically).

```
df[['total_votes']].plot(kind='hist')
```

```
<Axes: ylabel='Frequency'>
```



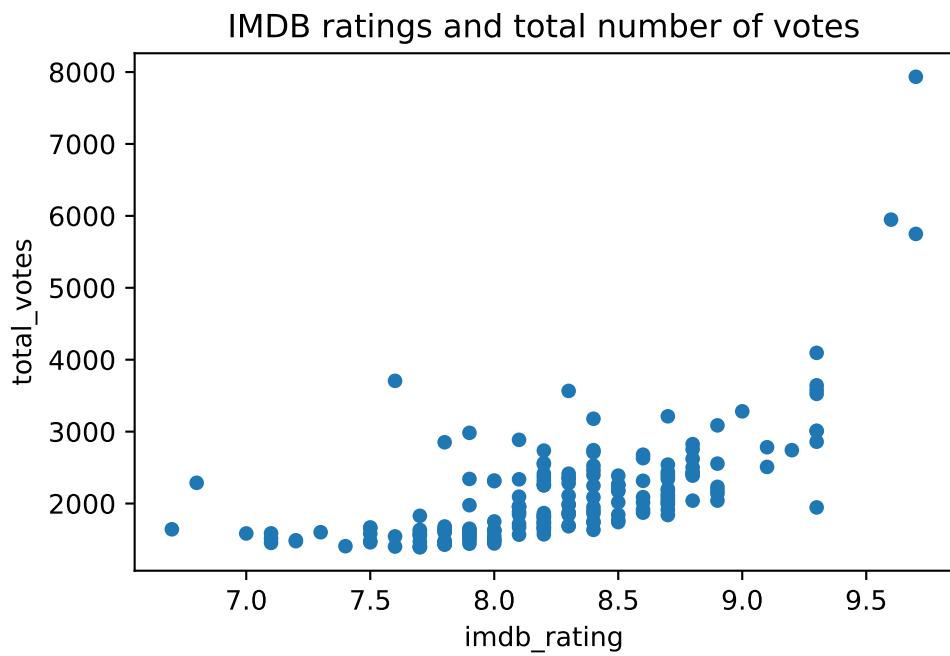
A positively skewed distribution - many smaller values and very few high values.

11.3 Bivariate

The number of votes and the imdb rating are not independent events. These two data variables are related.

Scatter plots are simple ways to explore the relationship between two data variables. Note, I use the term data variables instead of just variables to avoid any confusion.

```
df.plot(x='imdb_rating', y='total_votes', kind='scatter', title='IMDB ratings and total nu  
<Axes: title={'center': 'IMDB ratings and total number of votes'}, xlabel='imdb_rating', yla
```



That is really interesting. The episodes with the highest rating also have the greatest number of votes. There was clearly a great outpouring of happiness there.

Which episodes were they?

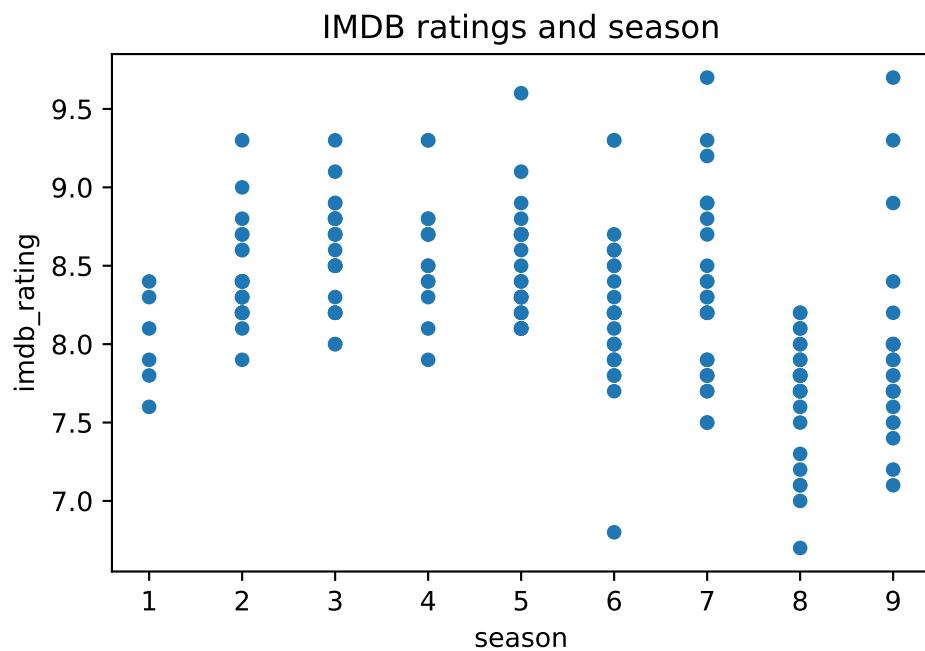
```
df[df['total_votes'] > 5000]
```

	season	episode	title	imdb_rating	total_votes	air_date
77	5	13	Stress Relief	9.6	5948	2009-02-01
137	7	21	Goodbye, Michael	9.7	5749	2011-04-28
187	9	23	Finale	9.7	7934	2013-05-16

Excellent. Any influence of season on votes?

```
df.plot(x='season', y='imdb_rating', kind='scatter', title='IMDB ratings and season')
```

```
<Axes: title={'center': 'IMDB ratings and season'}, xlabel='season', ylabel='imdb_rating'>
```



Season 8 seems to be a bit low. But nothing too extreme.

11.4 Dates

Our data contains air date information. Currently, that column is 'object' or a string.

```
df.head()
```

	season	episode	title	imdb_rating	total_votes	air_date
0	1	1	Pilot	7.6	3706	2005-03-24
1	1	2	Diversity Day	8.3	3566	2005-03-29
2	1	3	Health Care	7.9	2983	2005-04-05
3	1	4	The Alliance	8.1	2886	2005-04-12
4	1	5	Basketball	8.4	3179	2005-04-19

```
df.dtypes
```

```
season          int64
episode         int64
title           object
imdb_rating     float64
total_votes     int64
air_date        object
dtype: object
```

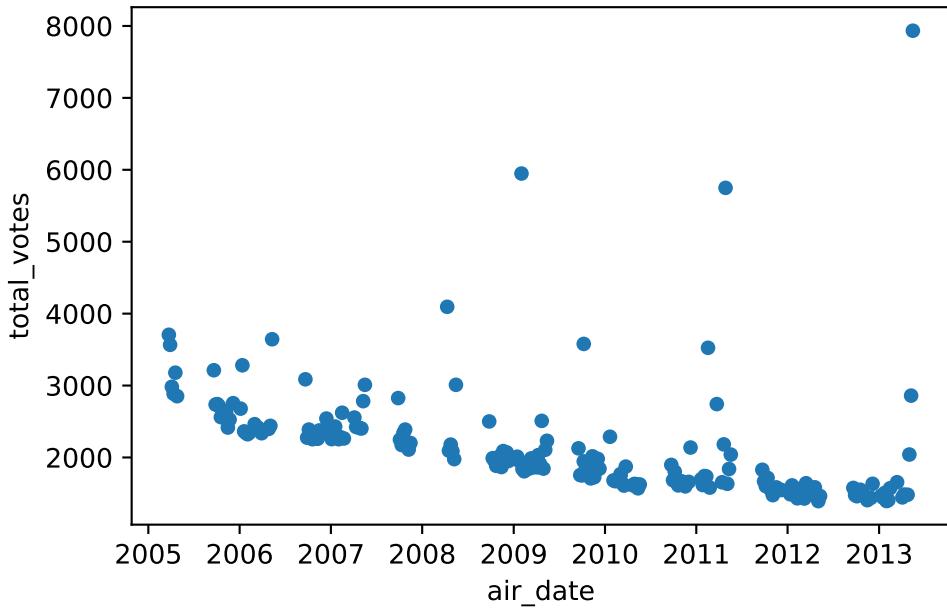
We can set this to be datetime instead. That will help us plot the time series of the data.

```
df['air_date'] = pd.to_datetime(df['air_date'])
df.dtypes
```

```
season          int64
episode         int64
title           object
imdb_rating     float64
total_votes     int64
air_date        datetime64[ns]
dtype: object
```

```
df.plot(x = 'air_date', y = 'total_votes', kind='scatter')
```

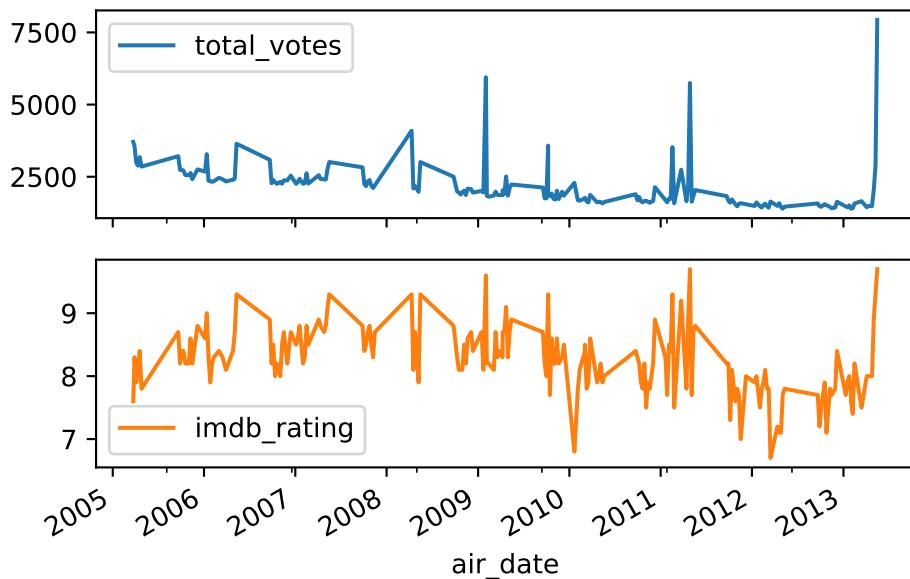
```
<Axes: xlabel='air_date', ylabel='total_votes'>
```



We can look at multiple variables using subplots.

```
df[['air_date', 'total_votes', 'imdb_rating']].plot(x = 'air_date', subplots=True)

array([<Axes: xlabel='air_date'>, <Axes: xlabel='air_date'>], dtype=object)
```

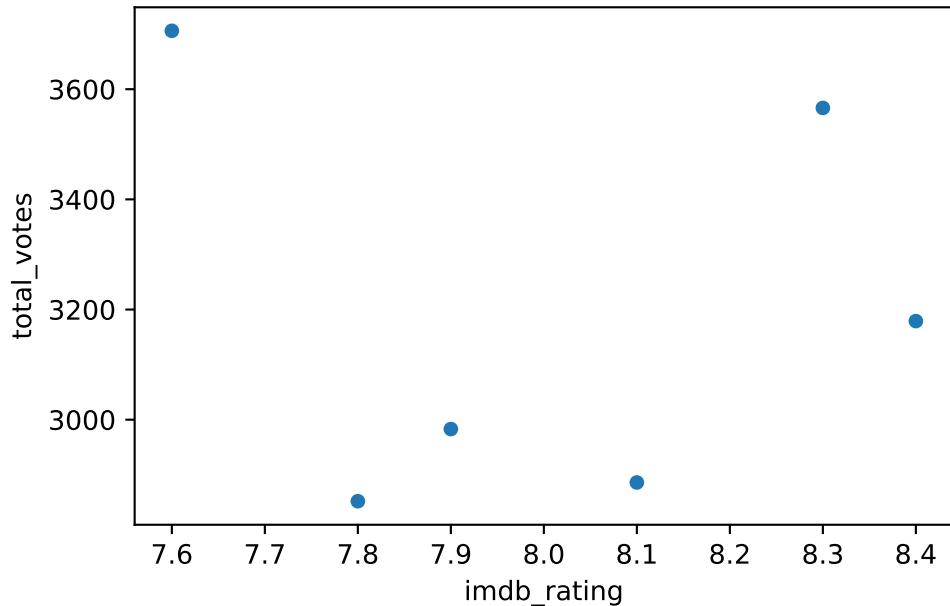


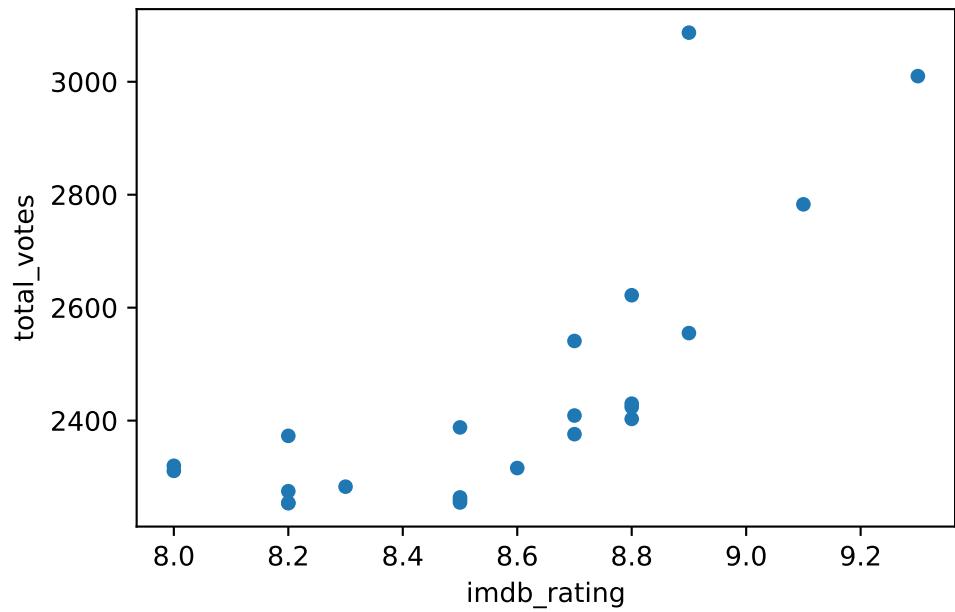
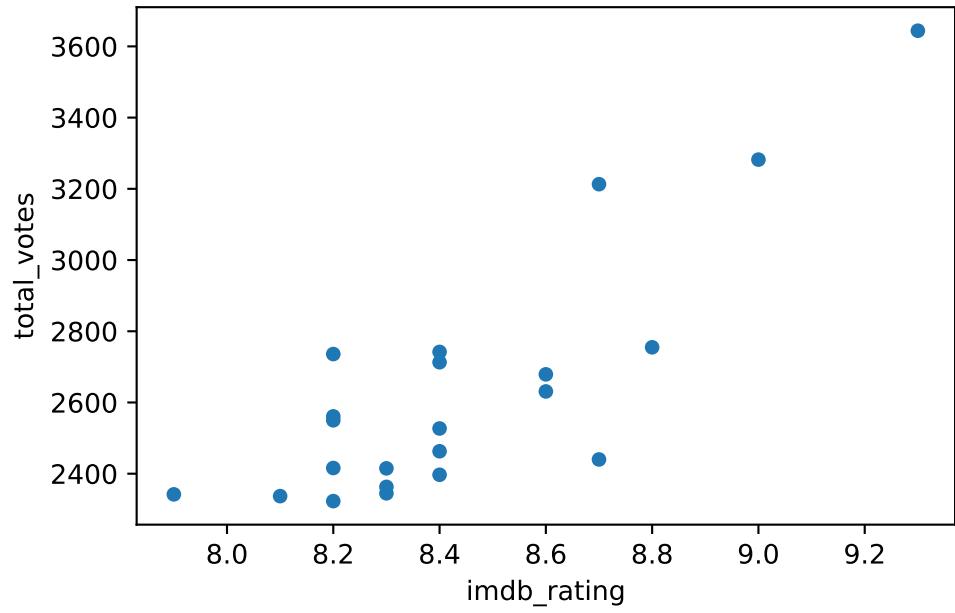
11.5 Multivariate

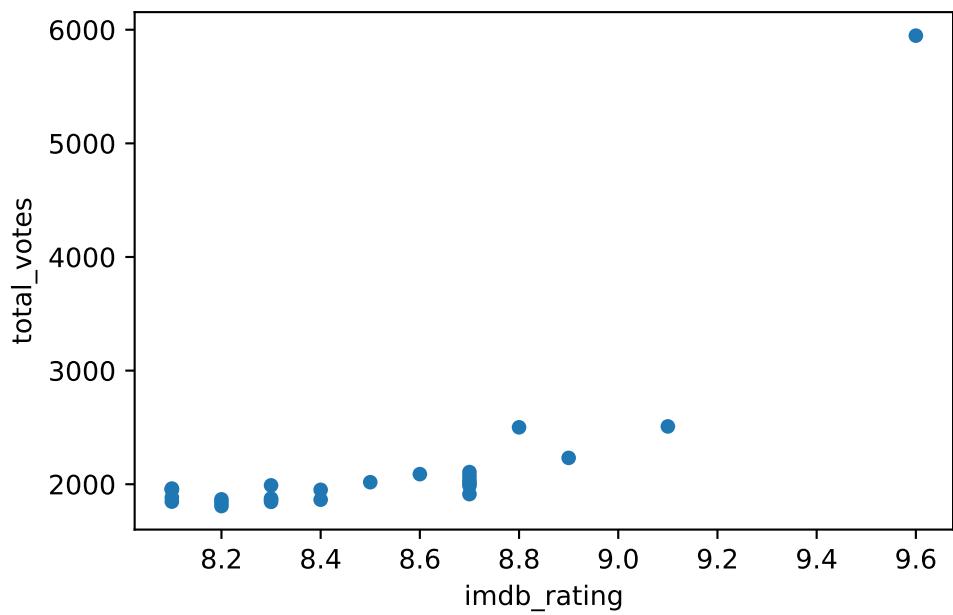
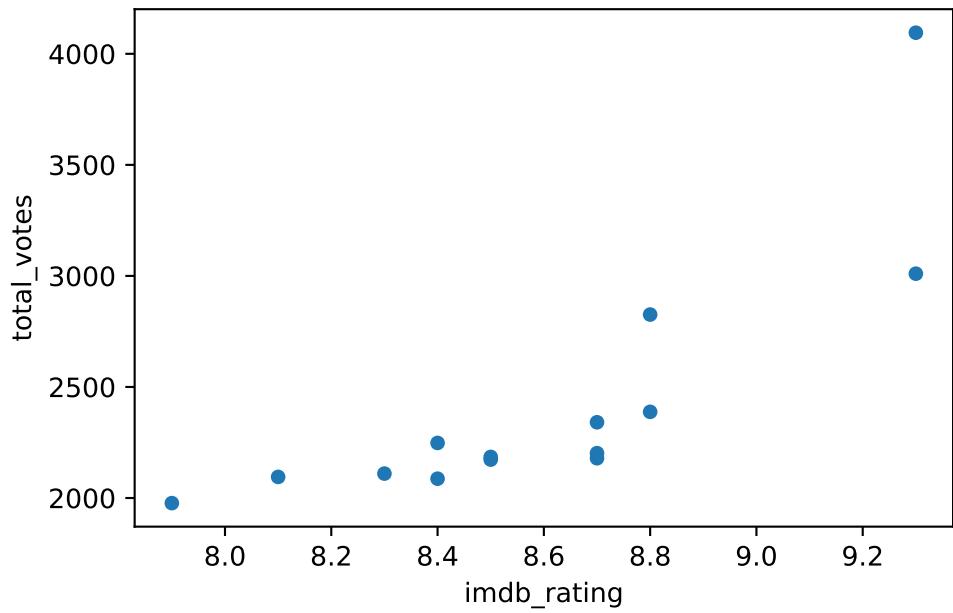
Our dataset is quite simple. But we can look at two variables (total_votes, imdb_rating) by a third (season).

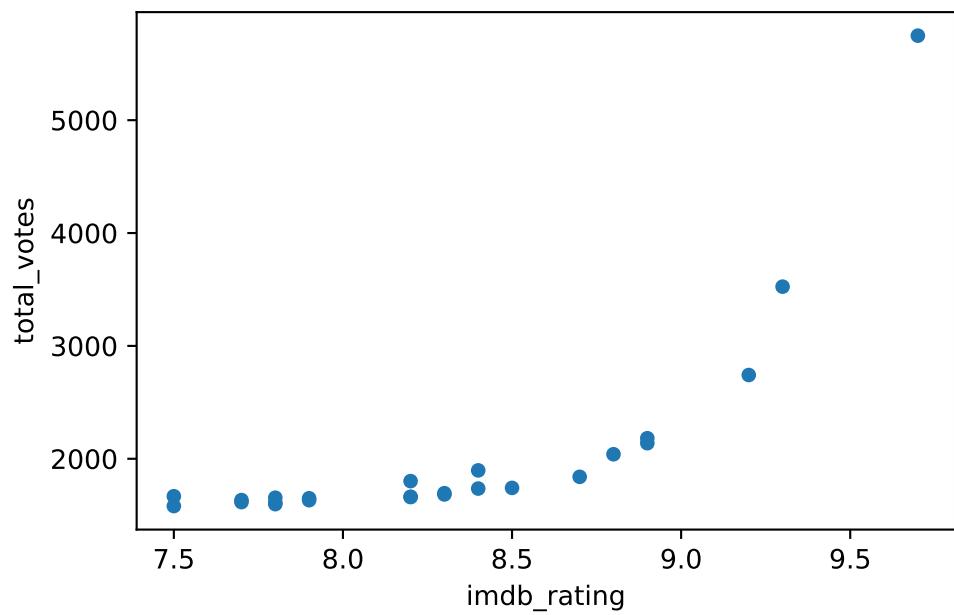
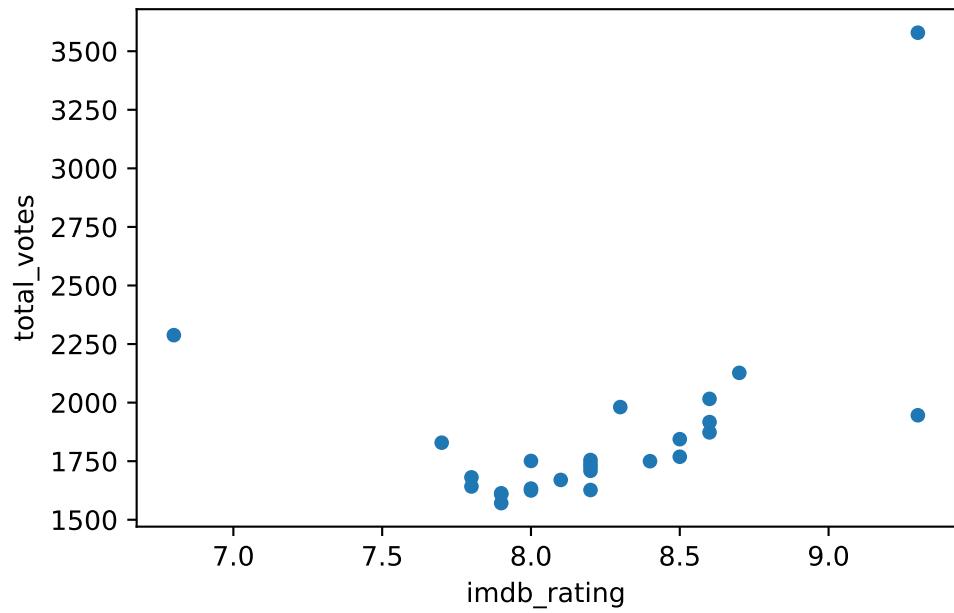
```
df.groupby('season').plot(kind='scatter', y = 'total_votes', x = 'imdb_rating')
```

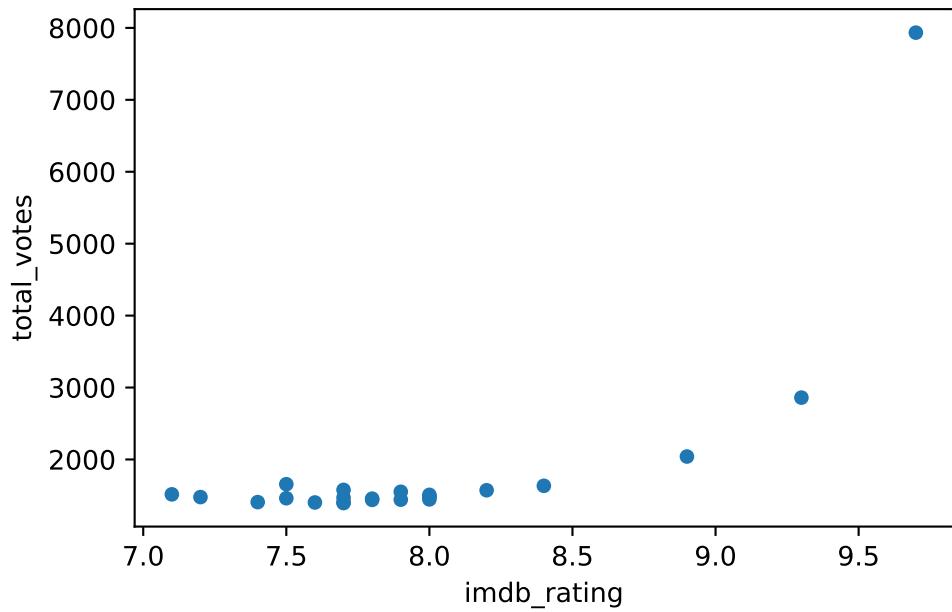
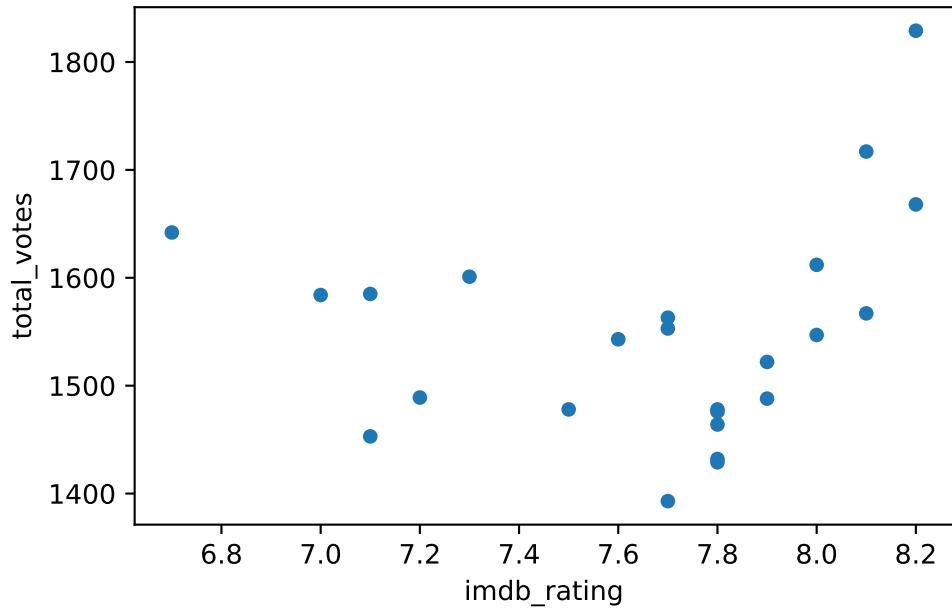
```
season
1    Axes(0.125,0.11;0.775x0.77)
2    Axes(0.125,0.11;0.775x0.77)
3    Axes(0.125,0.11;0.775x0.77)
4    Axes(0.125,0.11;0.775x0.77)
5    Axes(0.125,0.11;0.775x0.77)
6    Axes(0.125,0.11;0.775x0.77)
7    Axes(0.125,0.11;0.775x0.77)
8    Axes(0.125,0.11;0.775x0.77)
9    Axes(0.125,0.11;0.775x0.77)
dtype: object
```











There is a lot more you can do with plots with Pandas and Matplotlib. A good resource is the [visualisation section of the pandas documentation](#).

12 IM939 Lab 2 - Part 3

Our data can contain missing values or benefit from transformations.

12.1 Missing values

I have removed some of the ratings data in the office_ratings_missing.csv.

```
import pandas as pd
df = pd.read_csv('data/raw/office_ratings_missing.csv', encoding = 'UTF-8')

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 188 entries, 0 to 187
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
---  --  
 0   season      188 non-null    int64  
 1   episode     188 non-null    int64  
 2   title       188 non-null    object  
 3   imdb_rating 170 non-null    float64 
 4   total_votes 168 non-null    float64 
 5   air_date    188 non-null    object  
dtypes: float64(2), int64(2), object(2)
memory usage: 8.9+ KB
```

We are missing values in our imdb_rating and total votes columns.

```
df.shape[0] - df.count()

season          0
episode         0
```

```
title          0
imdb_rating    18
total_votes    20
air_date       0
dtype: int64
```

What to do? A quick solution is to either 0 the values or give them a roughly central value (the mean).

To do this we use the fillna method.

```
df['imdb_rating_with_0'] = df['imdb_rating'].fillna(0)
```

To fill with the mean.

```
df['imdb_rating_with_mean'] = df['imdb_rating'].fillna(df['imdb_rating'].mean())
```

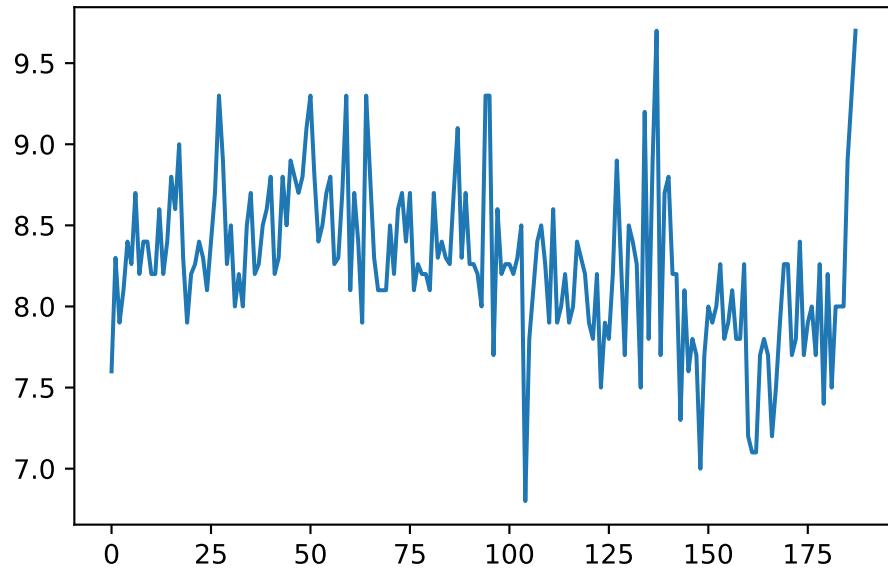
```
df.head()
```

	season	episode	title	imdb_rating	total_votes	air_date	imdb_rating_with_0	imdb
0	1	1	Pilot	7.6	NaN	24/03/2005	7.6	7.6
1	1	2	Diversity Day	8.3	3566.0	29/03/2005	8.3	8.3
2	1	3	Health Care	7.9	2983.0	05/04/2005	7.9	7.9
3	1	4	The Alliance	8.1	2886.0	12/04/2005	8.1	8.1
4	1	5	Basketball	8.4	3179.0	19/04/2005	8.4	8.4

We can plot these to see what looks most reasonable (you can probably also make an educated guess here).

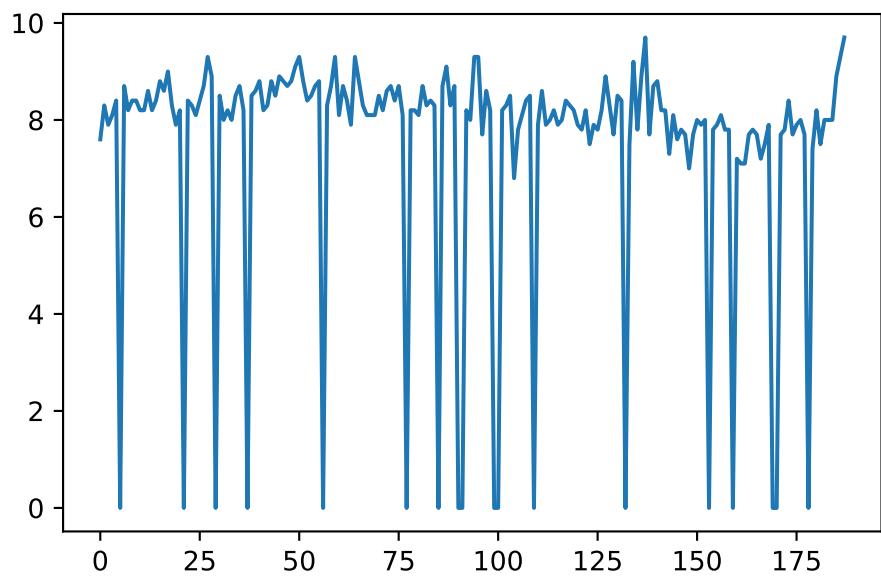
```
df['imdb_rating_with_mean'].plot()
```

```
<Axes: >
```



```
df['imdb_rating_with_0'].plot()
```

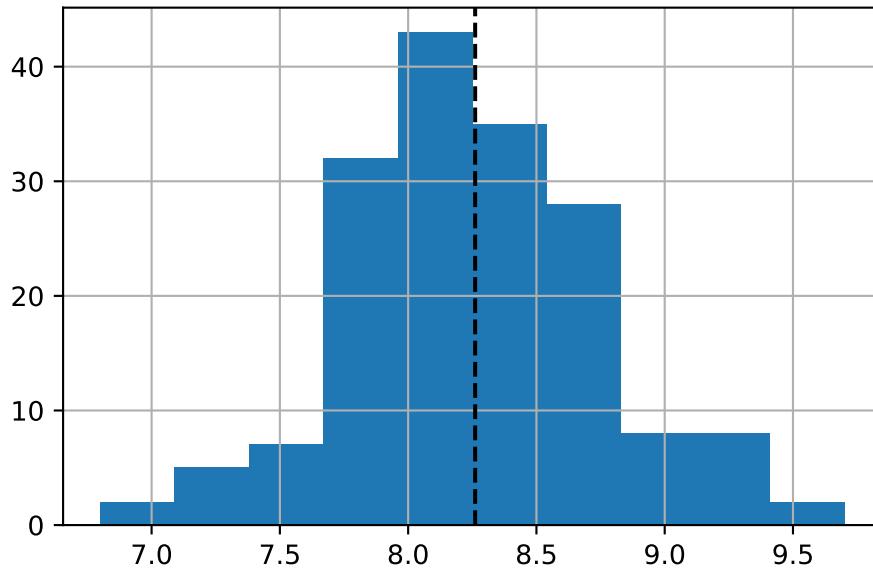
```
<Axes: >
```



Going with the mean seems quite sensible in this case. Especially as the data is gaussian so the mean is probably an accurate representation of the central value.

```
ax = df['imdb_rating'].hist()  
ax.axvline(df['imdb_rating'].mean(), color='k', linestyle='--')
```

```
<matplotlib.lines.Line2D at 0x1561bb9d0>
```



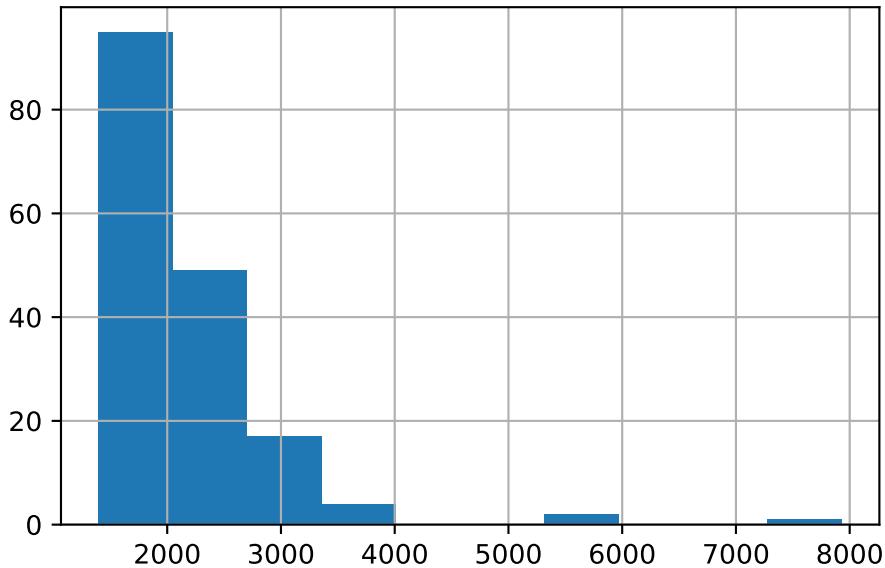
12.2 Transformations

Some statistical models, such as standard linear regression, require the predicted variable to be gaussian distributed (a single central point and a roughly symmetrical decrease in frequency, see [this Wolfram alpha page](#).

The distribution of votes is positively skewed (most values are low).

```
df['total_votes'].hist()
```

```
<Axes: >
```



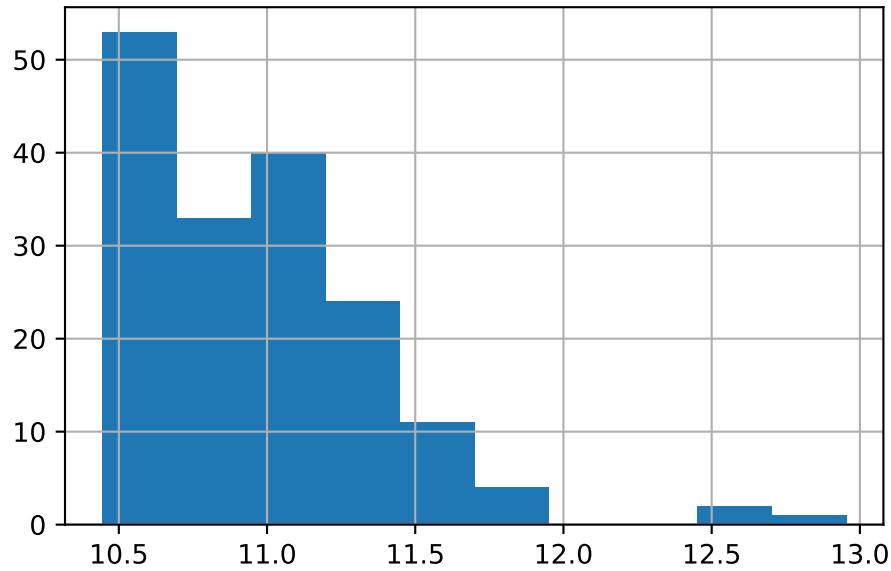
A log transformation can make this data closer to a gaussian distributed data variable. For the log transformation we are going to use numpy (numerical python) which is a rather excellent library.

```
import numpy as np

df['total_votes_log'] = np.log2(df['total_votes'])

df['total_votes_log'].hist()

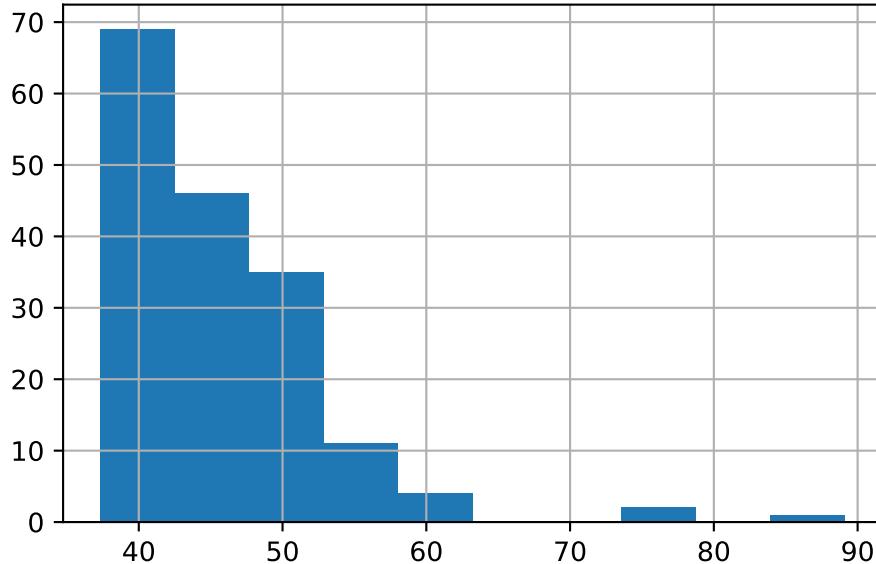
<Axes: >
```



That is less skewed, but not ideal. Perhaps a square root transformation instead?

```
df['total_votes_sqrt'] = np.sqrt(df['total_votes'])
df['total_votes_sqrt'].hist()
```

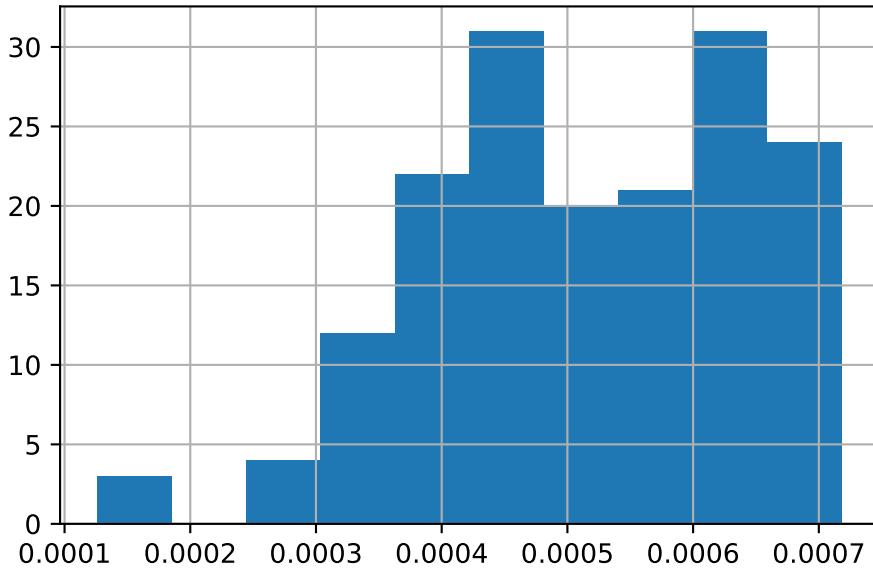
<Axes: >



...well, maybe a inverse/reciprocal transformation. It is possible we have hit the limit on what we can do.

```
df['total_votes_recip'] = np.reciprocal(df['total_votes'])
df['total_votes_recip'].hist()
```

<Axes: >



At this point, I think we should conceded that we can make the distribution less positively skewed. However, transformation are not magic and we cannot turn a heavily positively skewed distribution into a normally distributed one.

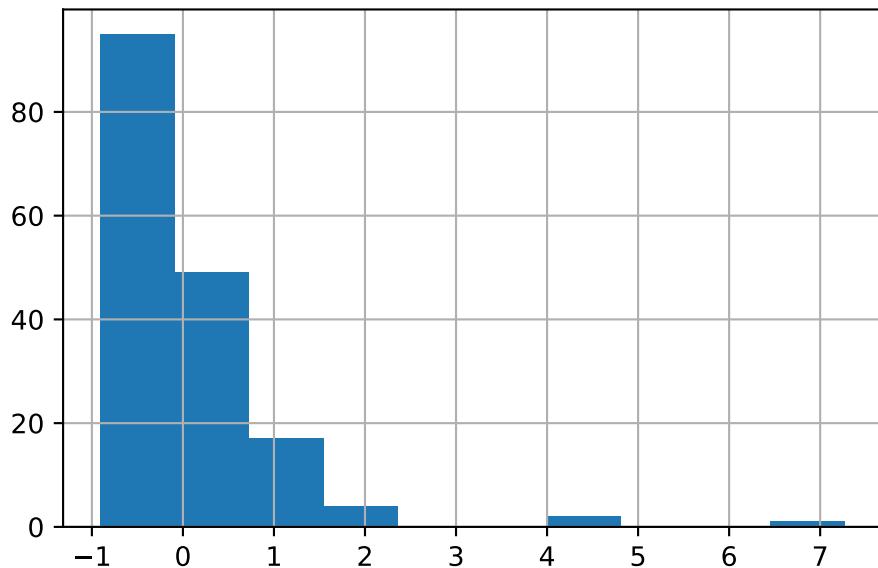
Oh well.

We can calculate z scores though so we can plot both total_votes and imdb_ratings on a single plot. Currently, the IMDB scores vary between 0 and 10 whereas the number of votes number in the thousands.

```
df['total_votes_z'] = (df['total_votes'] - df['total_votes'].mean()) / df['total_votes'].std()
df['imdb_rating_z'] = (df['imdb_rating'] - df['imdb_rating'].mean()) / df['imdb_rating'].std()

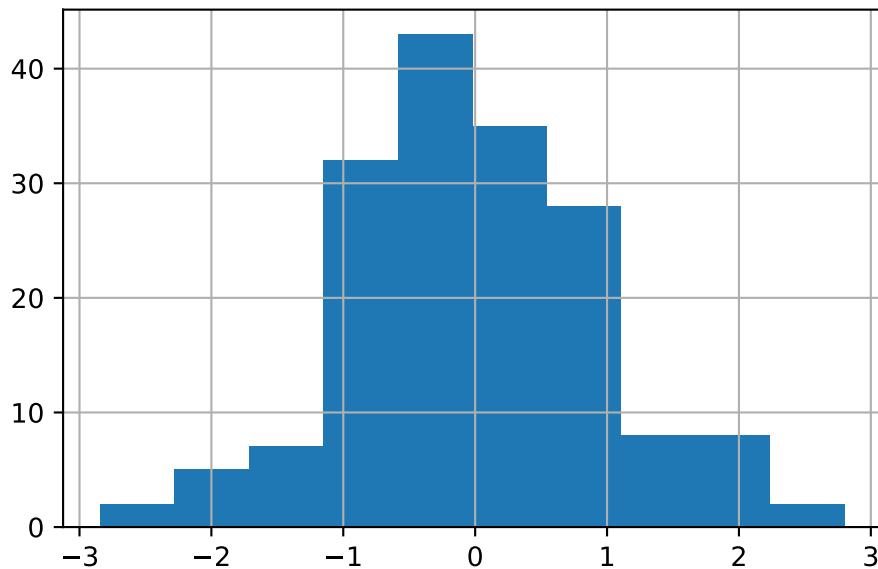
df['total_votes_z'].hist()
```

```
<Axes: >
```



```
df['imdb_rating_z'].hist()
```

```
<Axes: >
```



Now we can compare the trends in score and number of votes on a single plot.

We are going to use a slightly different approach to creating the plots. Called to the plot() method from Pandas actually use a library called matplotlib. We are going to use the pyplot module of matplotlib directly.

```
import matplotlib.pyplot as plt
```

Convert the air date into a datetime object.

```
df['air_date'] = pd.to_datetime(df['air_date'])
```

```
/var/folders/7v/z19mv52s3ls94kntlt_19ryh0000gq/T/ipykernel_54113/2716201442.py:1: UserWarning: All methods of the current Matplotlib backend (TkAgg) are deprecated and will be removed in a future version. You can use plt.style.use('ggplot') to switch to a backend that supports the newer API.
```

```
df['air_date'] = pd.to_datetime(df['air_date'])
```

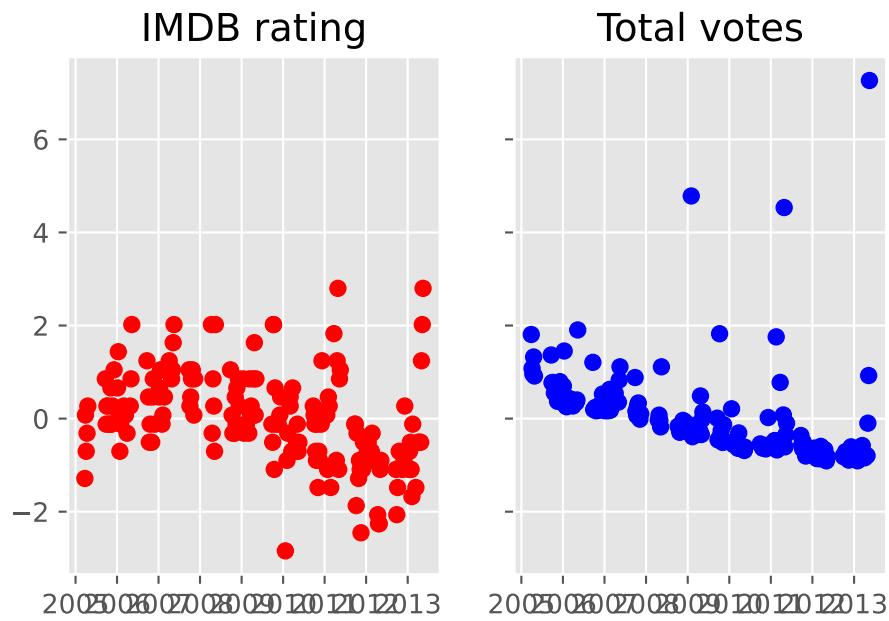
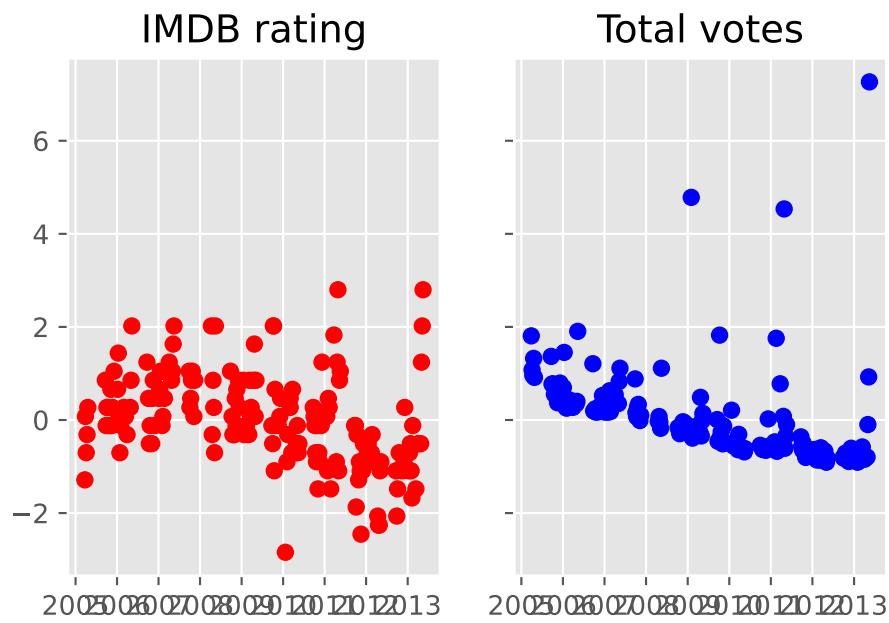
Then call the subplots function from pyplot to create two plots. From this we take the two plot axis (ax1, ax2) and call the method scatter for each to plot imdb_rating_z and total_votes_z.

```
plt.style.use('ggplot')

f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
ax1.scatter(df['air_date'], df['imdb_rating_z'], color = 'red')
ax1.set_title('IMDB rating')
ax2.scatter(df['air_date'], df['total_votes_z'], color = 'blue')
ax2.set_title('Total votes')

Text(0.5, 1.0, 'Total votes')
```

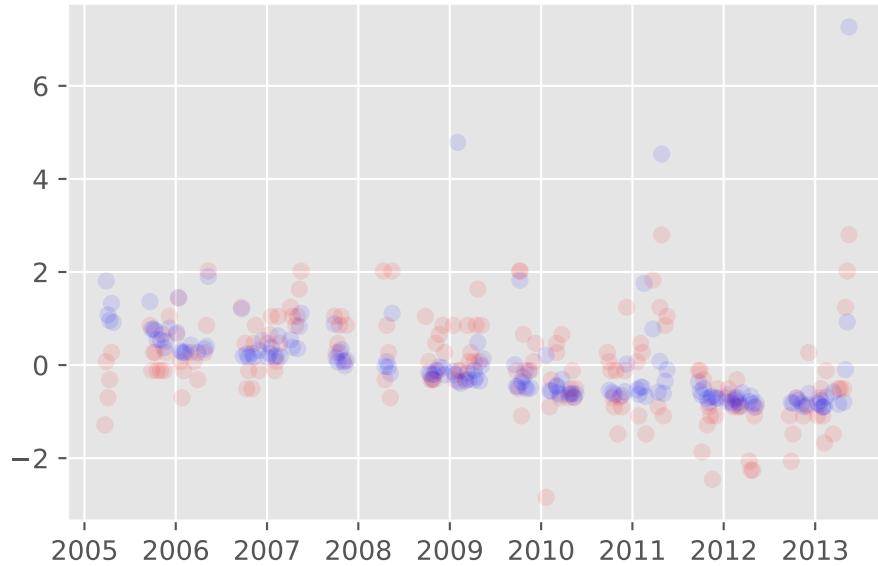
f



We can do better than that.

```
plt.scatter(df['air_date'], df['imdb_rating_z'], color = 'red', alpha = 0.1)
plt.scatter(df['air_date'], df['total_votes_z'], color = 'blue', alpha = 0.1)
```

```
<matplotlib.collections.PathCollection at 0x1566bd250>
```



```
?plt.scatter
```

We have done a lot so far. Exploring data in part 1, plotting data with the inbuilt Pandas methods in part 2 and dealing with both missing data and transformations in part 3.

In part 4, we will look at creating your own functions, a plotting library called seaborn and introduce a larger dataset.

13 IM939 Lab 2 - Part 4

13.1 Seaborn

Seaborn is another plotting library. Some consider it the ggplot of Python with excellent default setting which make your data life easier. There is rather good [documentation online](#) and it comes with Anaconda Python.

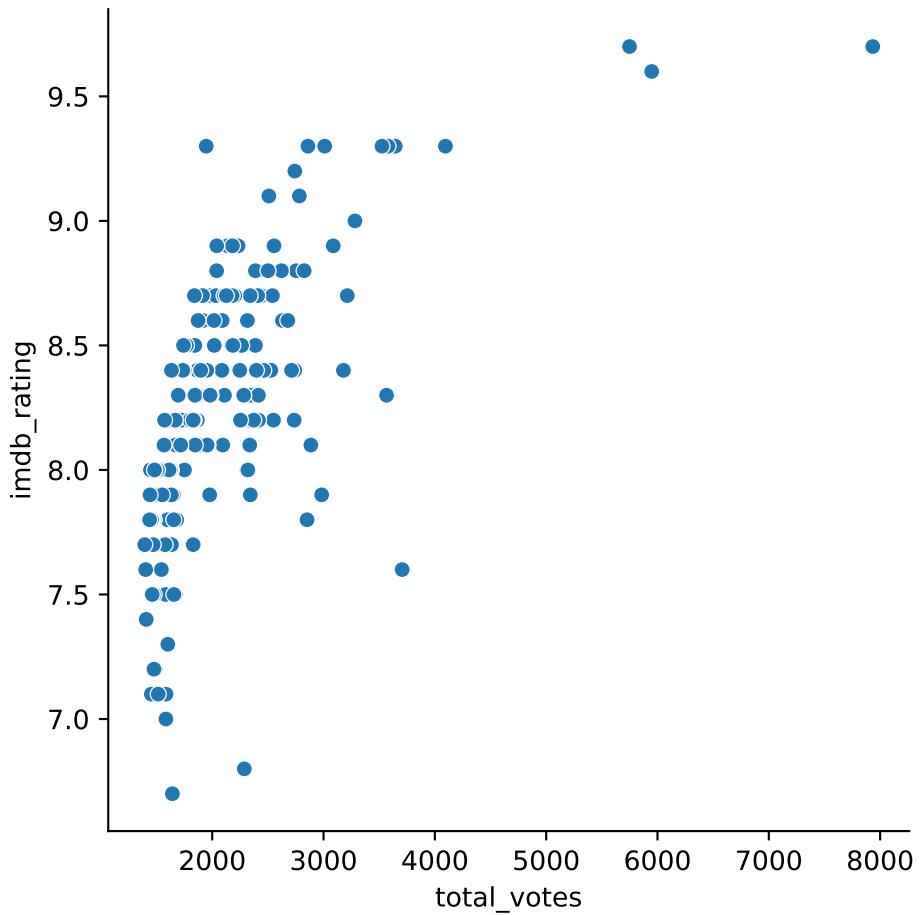
```
import numpy as np
import pandas as pd
import seaborn as sns
```

We can load and create the same plots as before.

```
office_df = pd.read_csv('data/raw/office_ratings.csv', encoding='UTF-8')
office_df.head()
```

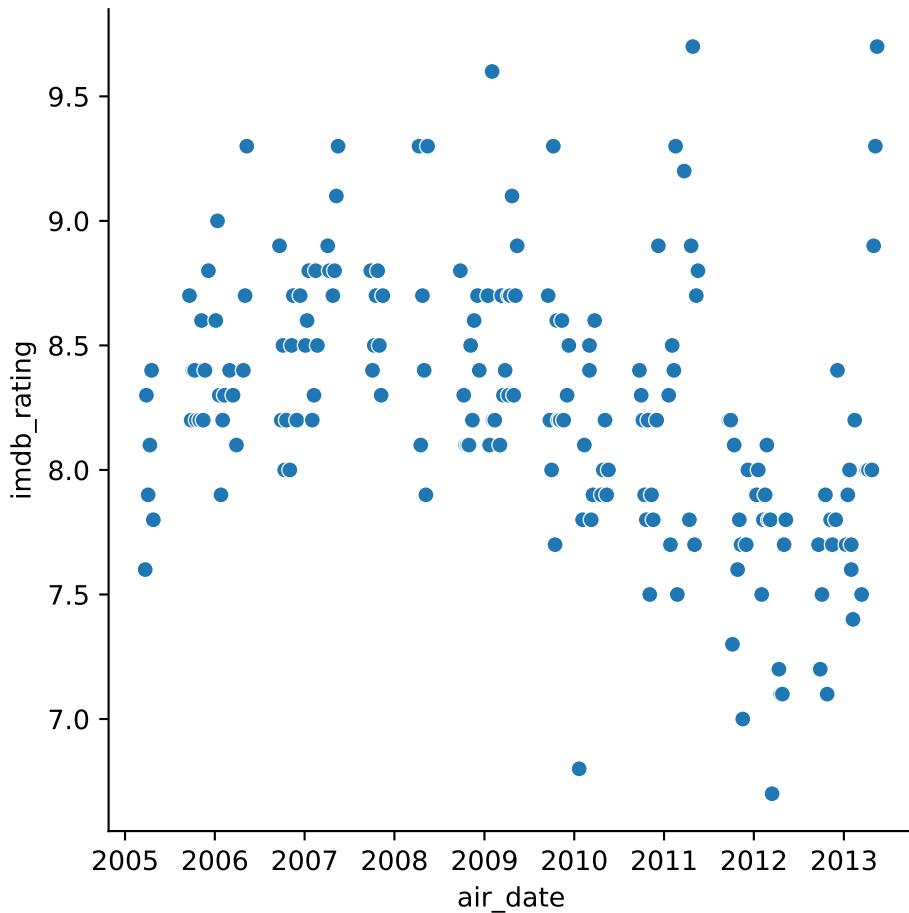
	season	episode	title	imdb_rating	total_votes	air_date
0	1	1	Pilot	7.6	3706	2005-03-24
1	1	2	Diversity Day	8.3	3566	2005-03-29
2	1	3	Health Care	7.9	2983	2005-04-05
3	1	4	The Alliance	8.1	2886	2005-04-12
4	1	5	Basketball	8.4	3179	2005-04-19

```
sns.relplot(x='total_votes', y='imdb_rating', data=office_df)
```



```
office_df['air_date'] = pd.to_datetime(office_df['air_date'], errors='ignore')

g = sns.relplot(x="air_date", y="imdb_rating", kind="scatter", data=office_df)
```



13.2 Functions

We can define our own functions. A function helps us with code we are going to run multiple times. For instance, the below function scales values between 0 and 1.

Here is a modified function from [stackoverflow](#).

```
office_df.head()
```

	season	episode	title	imdb_rating	total_votes	air_date
0	1	1	Pilot	7.6	3706	2005-03-24
1	1	2	Diversity Day	8.3	3566	2005-03-29
2	1	3	Health Care	7.9	2983	2005-04-05

	season	episode	title	imdb_rating	total_votes	air_date
3	1	4	The Alliance	8.1	2886	2005-04-12
4	1	5	Basketball	8.4	3179	2005-04-19

```
def normalize(df, feature_name):
    result = df.copy()

    max_value = df[feature_name].max()
    min_value = df[feature_name].min()

    result[feature_name] = (df[feature_name] - min_value) / (max_value - min_value)

    return result
```

Passing the dataframe and name of the column will return a dataframe with that column scaled between 0 and 1.

```
normalize(office_df, 'imdb_rating')
```

	season	episode	title	imdb_rating	total_votes	air_date
0	1	1	Pilot	0.300000	3706	2005-03-24
1	1	2	Diversity Day	0.533333	3566	2005-03-29
2	1	3	Health Care	0.400000	2983	2005-04-05
3	1	4	The Alliance	0.466667	2886	2005-04-12
4	1	5	Basketball	0.566667	3179	2005-04-19
...
183	9	19	Stairmageddon	0.433333	1484	2013-04-11
184	9	20	Paper Airplane	0.433333	1482	2013-04-25
185	9	21	Livin' the Dream	0.733333	2041	2013-05-02
186	9	22	A.A.R.M.	0.866667	2860	2013-05-09
187	9	23	Finale	1.000000	7934	2013-05-16

Replacing the original dataframe. We can normalize both out votes and rating.

```
office_df = normalize(office_df, 'imdb_rating')

office_df = normalize(office_df, 'total_votes')
```

```
office_df
```

	season	episode	title	imdb_rating	total_votes	air_date
0	1	1	Pilot	0.300000	0.353616	2005-03-24
1	1	2	Diversity Day	0.533333	0.332212	2005-03-29
2	1	3	Health Care	0.400000	0.243082	2005-04-05
3	1	4	The Alliance	0.466667	0.228253	2005-04-12
4	1	5	Basketball	0.566667	0.273047	2005-04-19
...
183	9	19	Stairmageddon	0.433333	0.013912	2013-04-11
184	9	20	Paper Airplane	0.433333	0.013606	2013-04-25
185	9	21	Livin' the Dream	0.733333	0.099067	2013-05-02
186	9	22	A.A.R.M.	0.866667	0.224278	2013-05-09
187	9	23	Finale	1.000000	1.000000	2013-05-16

Seaborn prefers a long format table. Details of melt can be found [here](#).

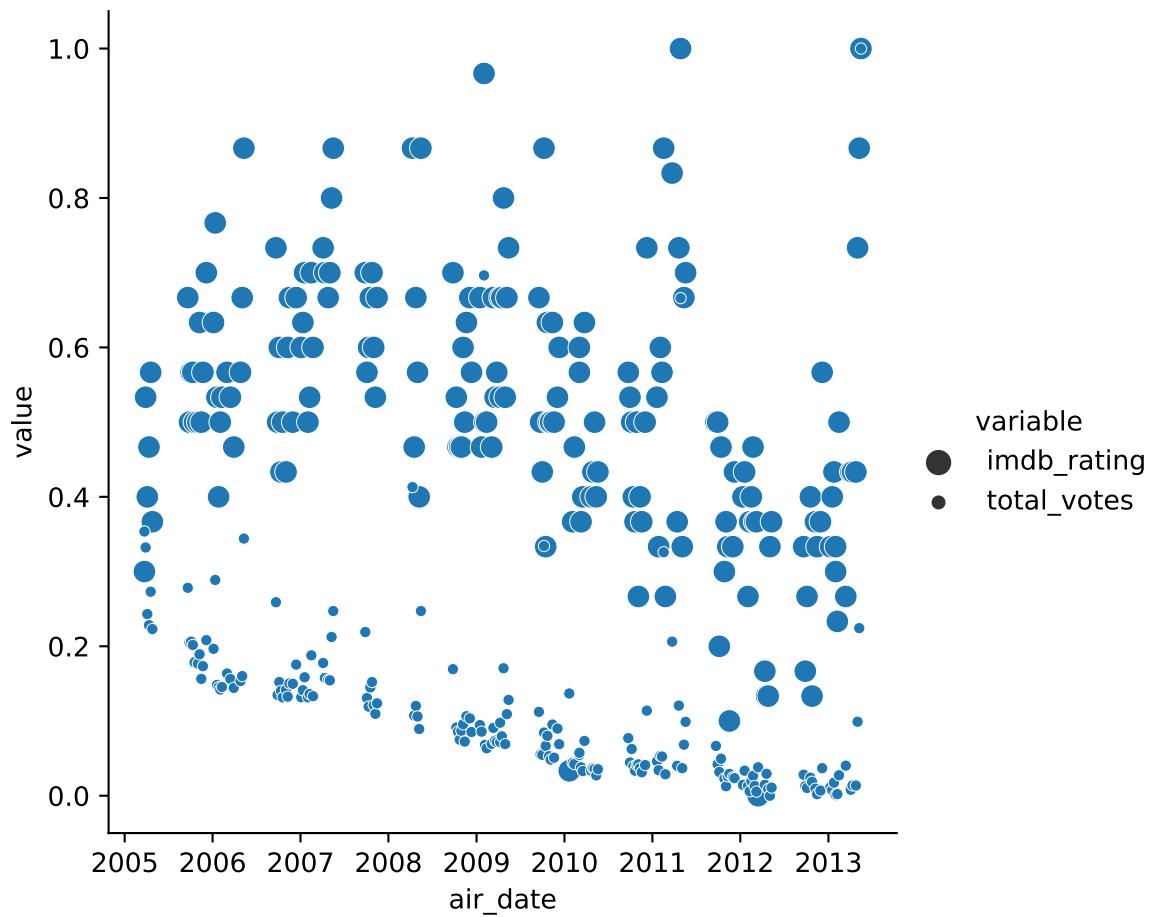
```
office_df_long=pd.melt(office_df, id_vars=['season', 'episode', 'title', 'air_date'], value
```

```
office_df_long
```

	season	episode	title	air_date	variable	value
0	1	1	Pilot	2005-03-24	imdb_rating	0.300000
1	1	2	Diversity Day	2005-03-29	imdb_rating	0.533333
2	1	3	Health Care	2005-04-05	imdb_rating	0.400000
3	1	4	The Alliance	2005-04-12	imdb_rating	0.466667
4	1	5	Basketball	2005-04-19	imdb_rating	0.566667
...
371	9	19	Stairmageddon	2013-04-11	total_votes	0.013912
372	9	20	Paper Airplane	2013-04-25	total_votes	0.013606
373	9	21	Livin' the Dream	2013-05-02	total_votes	0.099067
374	9	22	A.A.R.M.	2013-05-09	total_votes	0.224278
375	9	23	Finale	2013-05-16	total_votes	1.000000

Which we can plot in seaborn like so.

```
sns.relplot(x='air_date', y='value', size='variable', data=office_df_long)
```



```
?sns.relplot
```

Part IV

Abstractions & Models

14 Lab: Data Processing and Summarization

This Cagatay's code helps you go through different functions in data processing and summarization

```
# Importing libraries
import os
import numpy as np
import pandas as pd
import statsmodels.api as sm
from scipy.spatial.distance import cdist # This is the function that computes the Mahalanobis distance
from scipy.stats import skew, kurtosis # The skew and kurtosis of a distribution
from statsmodels.robust.scale import mad # This computes the median absolute deviation of a column
import matplotlib.pyplot as plt
%matplotlib inline
```

14.1 Part I: Outliers

- 1) Load the data on properties of cars into a pd dataframe

```
# Loading Data
df = pd.read_csv('data/raw/accord_sedan.csv')

# Inspecting the few first rows of the dataframe
df.head()
```

	price	mileage	year	trim	engine	transmission
0	14995	67697	2006	ex	4 Cyl	Manual
1	11988	73738	2006	ex	4 Cyl	Manual
2	11999	80313	2006	lx	4 Cyl	Automatic
3	12995	86096	2006	lx	4 Cyl	Automatic
4	11333	79607	2006	lx	4 Cyl	Automatic

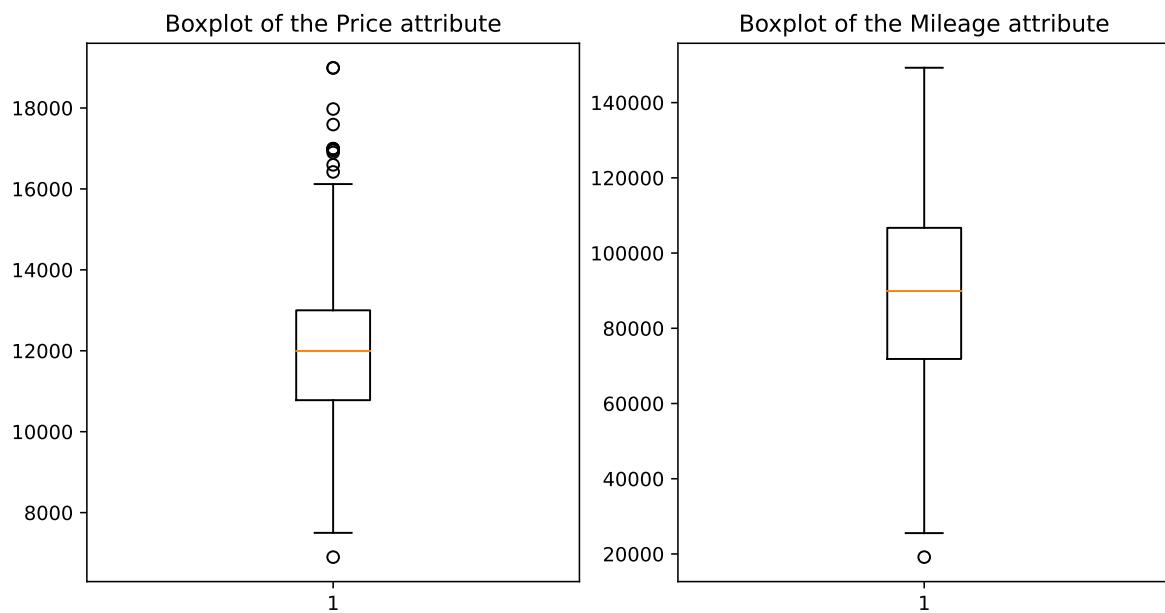
- 2) Visualise the columns: "price" and "mileage"

```

# 1D Visualizations
plt.figure(figsize=(10, 5))
plt.subplot(1,2,1)
plt.boxplot(df.price)
plt.title("Boxplot of the Price attribute")
plt.subplot(1,2,2)
plt.boxplot(df.mileage)
plt.title("Boxplot of the Mileage attribute");

#A semicolon in Python denotes separation, rather than termination.
#It allows you to write multiple statements on the same line. This

```



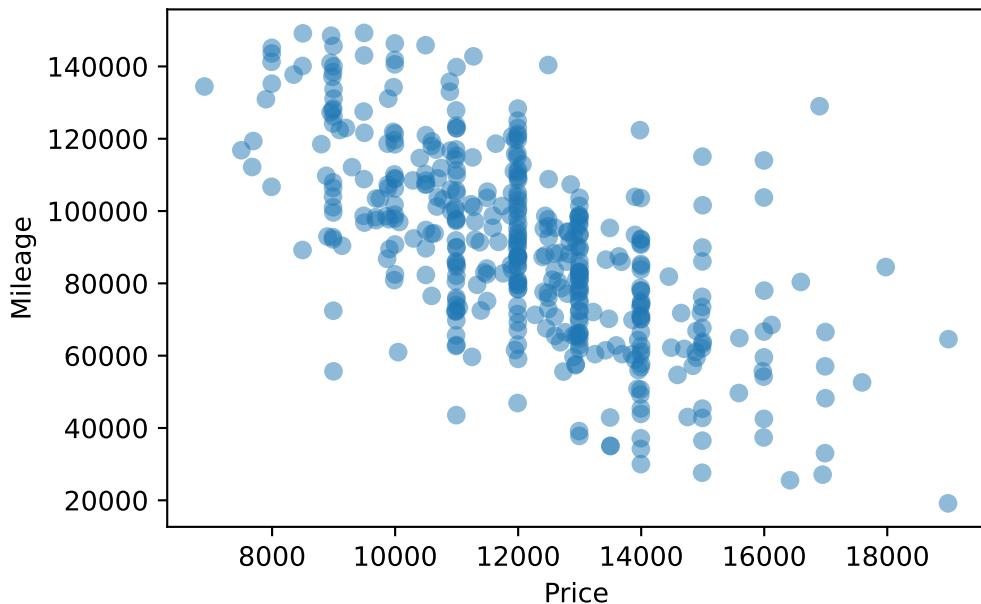
- 1) Identify the 2D outliers using the visualisation

```

# 2D Visualization
plt.scatter(df.price, df.mileage, alpha = .5)
plt.xlabel('Price')
plt.ylabel('Mileage')
plt.title('Mileage vs. Price\n');

```

Mileage vs. Price



Visually, outliers appear to be outside the ‘normal’ range of the rest of the points. A few outliers are quite obvious to spot, but the choice of the threshold (the limit after which you decide to label a point as an outlier) visually remains a very subjective matter.

- 4) Add two new columns to the dataframe called `isOutlierPrice` and `isOutlierMileage`. For the price column, calculate the mean and standard deviation. Find any rows that are more than 2 times standard deviations away from the mean and mark them with a 1 in the `isOutlierPrice` column. Do the same for mileage column

```
# Computing the isOutlierPrice column
upper_threshold_price = df.price.mean() + 2*df.price.std()
lower_threshold_price = df.price.mean() - 2*df.price.std()
df['isOutlierPrice'] = ((df.price > upper_threshold_price) | (df.price < lower_threshold_price))

# Computing the isOutlierMileage column
upper_threshold_mileage = df.mileage.mean() + 2*df.mileage.std()
lower_threshold_mileage = df.mileage.mean() - 2*df.mileage.std()
df['isOutlierMileage'] = ((df.mileage > upper_threshold_mileage) | (df.mileage < lower_threshold_mileage))

# Second way of doing the above using the np.where() function
#df['isOutlierPrice'] = np.where(abs(df.price - df.price.mean()) < 2*df.price.std(), False, True)
```

```
#df['isOutlierMileage'] = np.where(abs(df.mileage - df.price.mileage()) < 2*df.mileage.std)

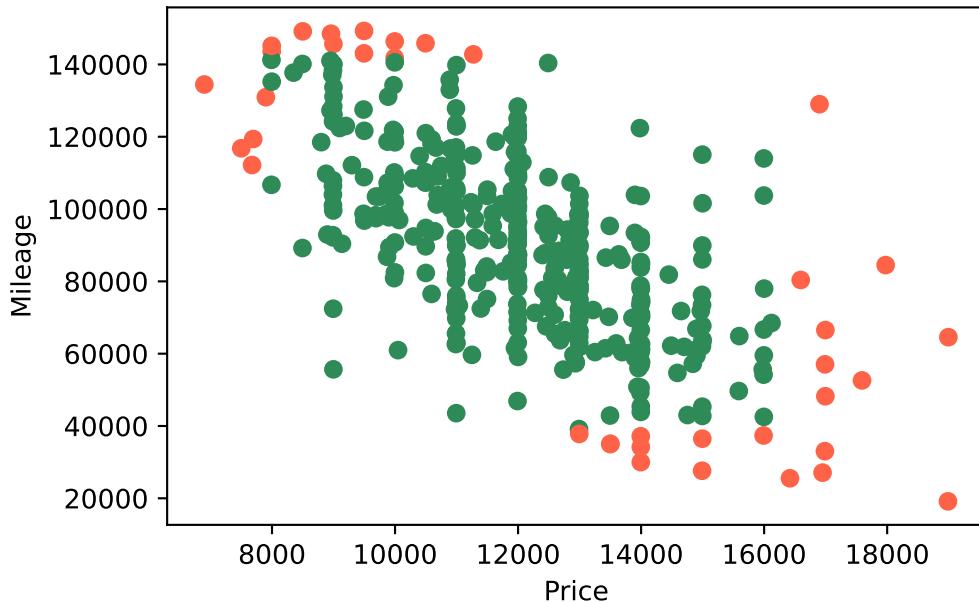
# Inspect the new DataFrame with the added columns
df.head()
```

	price	mileage	year	trim	engine	transmission	isOutlierPrice	isOutlierMileage
0	14995	67697	2006	ex	4 Cyl	Manual	False	False
1	11988	73738	2006	ex	4 Cyl	Manual	False	False
2	11999	80313	2006	lx	4 Cyl	Automatic	False	False
3	12995	86096	2006	lx	4 Cyl	Automatic	False	False
4	11333	79607	2006	lx	4 Cyl	Automatic	False	False

- 5) Visualize these values with a different color in the plot. Observe whether they are the same as you would mark them

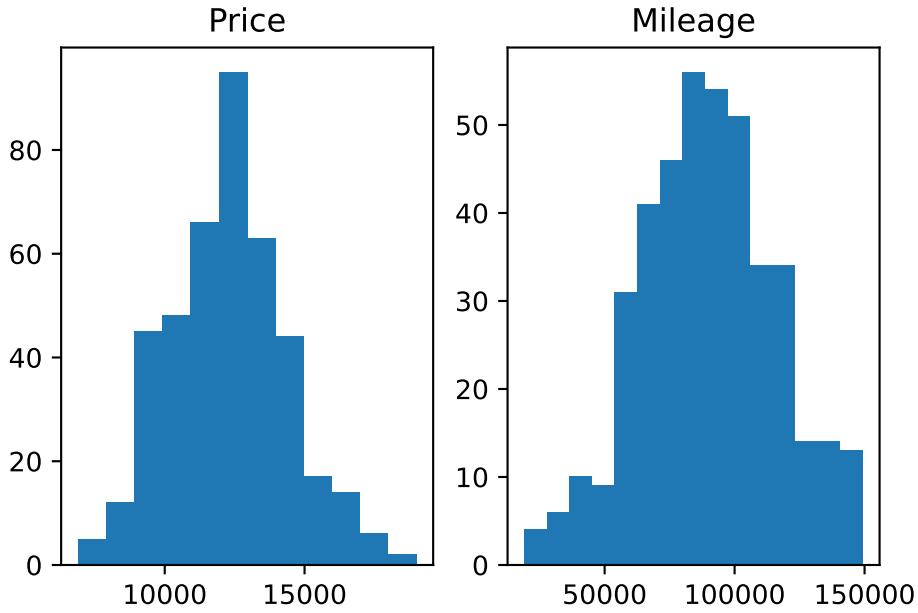
```
# Visualizing outliers in a different color
col = ['tomato' if i+j else 'seagreen' for i,j in zip(df.isOutlierPrice, df.isOutlierMileage)]
plt.scatter(df.price, df.mileage, color = col)
plt.xlabel('Price')
plt.ylabel('Mileage')
plt.title('Mileage vs. Price : Outliers 2+ std's away from the mean\n');
```

Mileage vs. Price : Outliers 2+ std's away from the mean



Visually filtering out outliers can be an effective tactic if we're just trying to conduct a quick and dirty experimentation. However, when we need to perform a solid and founded analysis, it's better to have a robust justification for our choices. In this case, we can use the deviation from the mean to define a threshold that separates 'normal' values from 'outliers'. Here, we opted for a two standard deviation threshold. The mathematical intuition behind this, is that under the normality assumption (if we assume our variable is normally distributed, which it almost is, refer to the next plot), then the probability of it having a value two standard deviations OR MORE away from the mean, is around 5%, which is very unlikely to happen. This is why we label these data points as outliers with respect to the (assumed) probability distribution of the variable. But this remains a way to identify 1D outliers only (identifying outliers within each column separately)

```
# Histograms of Price and Mileage (checking the normality assumption)
plt.subplot(1,2,1)
plt.hist(df.price, bins = 12)
plt.title('Price')
plt.subplot(1,2,2)
plt.hist(df.mileage, bins = 15)
plt.title('Mileage');
```



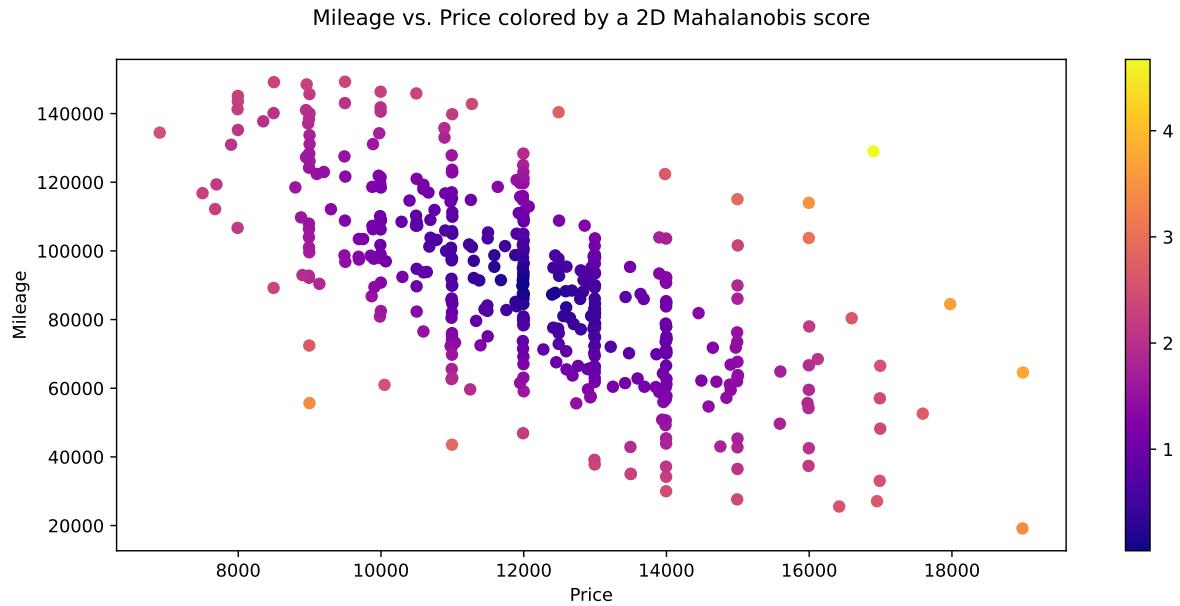
:)

- 6) Using the 2D Mahalanobis distance to find outliers

```
# Mean vector (computing the mean returns a Series, which we need to convert back to a DataFrame)
mean_v = df.iloc[:, 0:2].mean().to_frame().T # DataFrame.T returns the Transpose of the DataFrame
#mean_v = np.asarray([df.price.mean(), df.mileage.mean()]).reshape(1,2) # This is a better way of doing it

# Computing the Mahalanobis distance of each row to the mean vector
d = cdist(df.iloc[:, 0:2], mean_v, metric='mahalanobis')
#d = cdist(df[['price', 'mileage']].values, mean_v, metric='mahalanobis') # Another way of doing it

# Visualizing the scatter plot while coloring each point (i.e row) with a color from a chosen colormap
plt.figure(figsize=(12, 5))
plt.scatter(df.price, df.mileage, c = d.flatten(), cmap = 'plasma') # in order to know why we use plasma
plt.colorbar() # to show the colorbar
plt.xlabel('Price')
plt.ylabel('Mileage')
plt.title('Mileage vs. Price colored by a 2D Mahalanobis score\n');
```



14.2 Part II : Q-Q Plots

```
# Getting the Data in
df_tuber = pd.read_csv('data/raw/TB_burden_countries_2014-09-29.csv')

# Filling missing numeric values (I repeat NUMERIC COLUMNS, I didn't touch categorical ones)
df_tuber = df_tuber.fillna(value=df_tuber.mean(numeric_only = True))

# Inspecting missing values in dataset
pd.isnull(df_tuber).sum()
```

country	0
iso2	23
iso3	0
iso_numeric	0
g_whoregion	0
year	0
e_pop_num	0
e_prev_100k	0
e_prev_100k_lo	0
e_prev_100k_hi	0
e_prev_num	0

```

e_prev_num_lo          0
e_prev_num_hi          0
e_mort_exc_tbhiv_100k 0
e_mort_exc_tbhiv_100k_lo 0
e_mort_exc_tbhiv_100k_hi 0
e_mort_exc_tbhiv_num   0
e_mort_exc_tbhiv_num_lo 0
e_mort_exc_tbhiv_num_hi 0
source_mort            1
e_inc_100k              0
e_inc_100k_lo           0
e_inc_100k_hi           0
e_inc_num               0
e_inc_num_lo             0
e_inc_num_hi             0
e_tbhiv_prct            0
e_tbhiv_prct_lo          0
e_tbhiv_prct_hi          0
e_inc_tbhiv_100k          0
e_inc_tbhiv_100k_lo        0
e_inc_tbhiv_100k_hi        0
e_inc_tbhiv_num           0
e_inc_tbhiv_num_lo         0
e_inc_tbhiv_num_hi         0
source_tbhiv             1
c_cdr                   0
c_cdr_lo                 0
c_cdr_hi                 0
dtype: int64

```

- 1) Pick one of the columns from the Tuberculosis data and copy it into a numpy array as before

```

# Picking a column (I created a variable for this so I (and you (: ) can modify the column
colname = 'e_prev_100k'

# Creating a numpy array from our column
col = np.array(df_tuber[colname])

# Printing the type of our newly created column
print(type(col))

```

```
<class 'numpy.ndarray'>
```

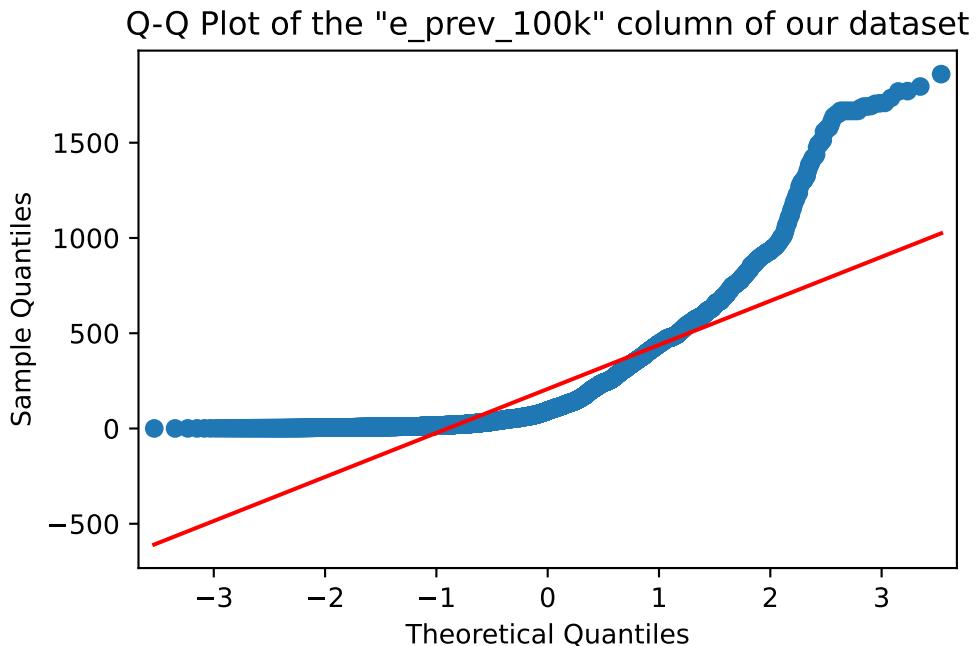
2) Compare this selected column to a Normal distribution. Then Sample from a Normal distribution and show a second Q-Q plot

```
# Plotting the Q-Q Plot for our column
sm.qqplot(col, line='r')
plt.title('Q-Q Plot of the "{}" column of our dataset'.format(colname));

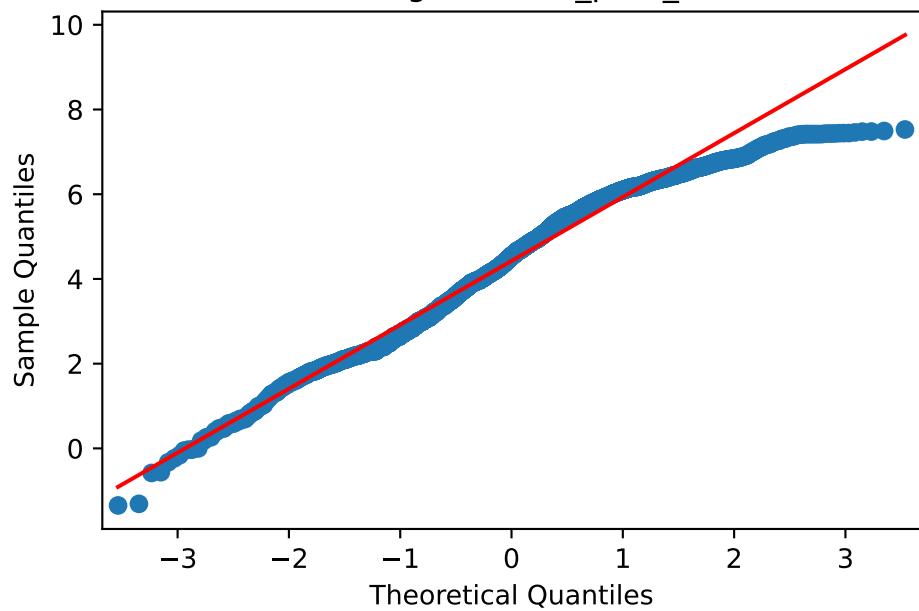
# Plotting the Q-Q Plot for the log of our column
sm.qqplot(np.log(col), line='r')
plt.title('Q-Q Plot of the Log of the "{}" column'.format(colname));

# Sampling from a Gaussian and a uniform distribution
sample_norm = np.random.randn(1000)
sample_unif = np.random.rand(1000)

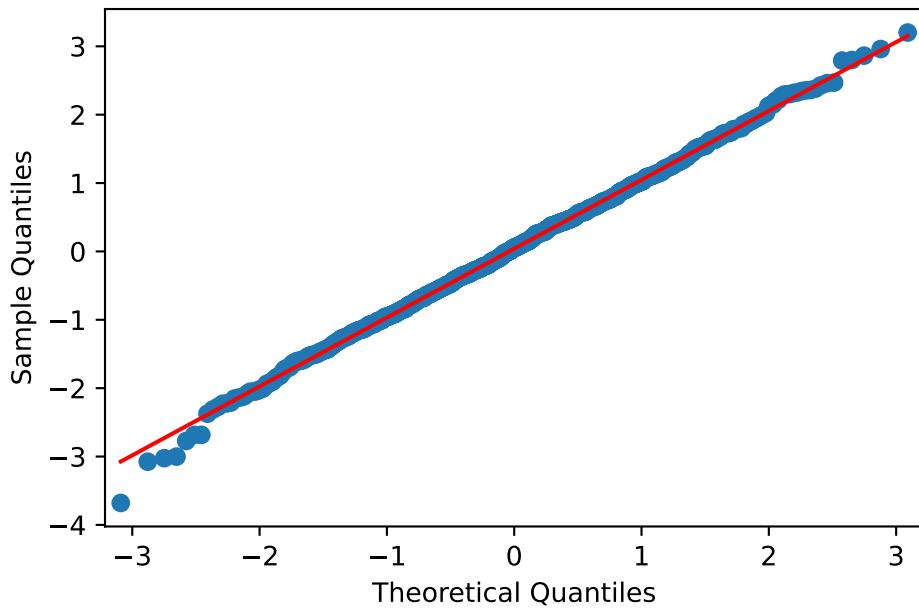
# Plotting the second Q-Q Plot for our sample (that was generated using a normal distribution)
sm.qqplot(sample_norm, line='r')
plt.title('Q-Q Plot of the generated sample (Gaussian)')
sm.qqplot(sample_unif, line='r')
plt.title('Q-Q Plot of the generated sample (Uniform)');
```

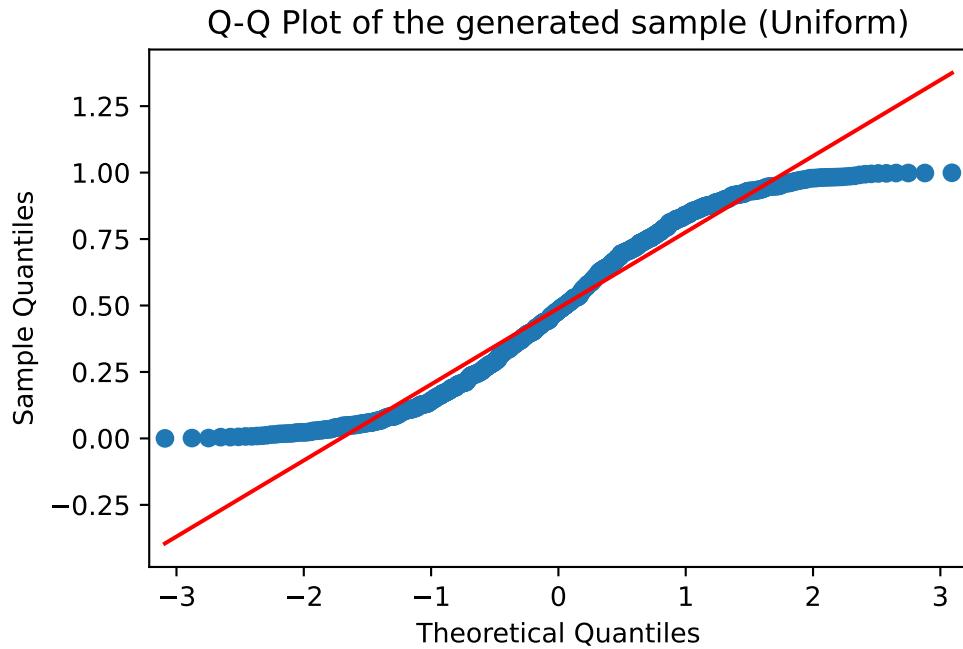


Q-Q Plot of the Log of the "e_prev_100k" column



Q-Q Plot of the generated sample (Gaussian)





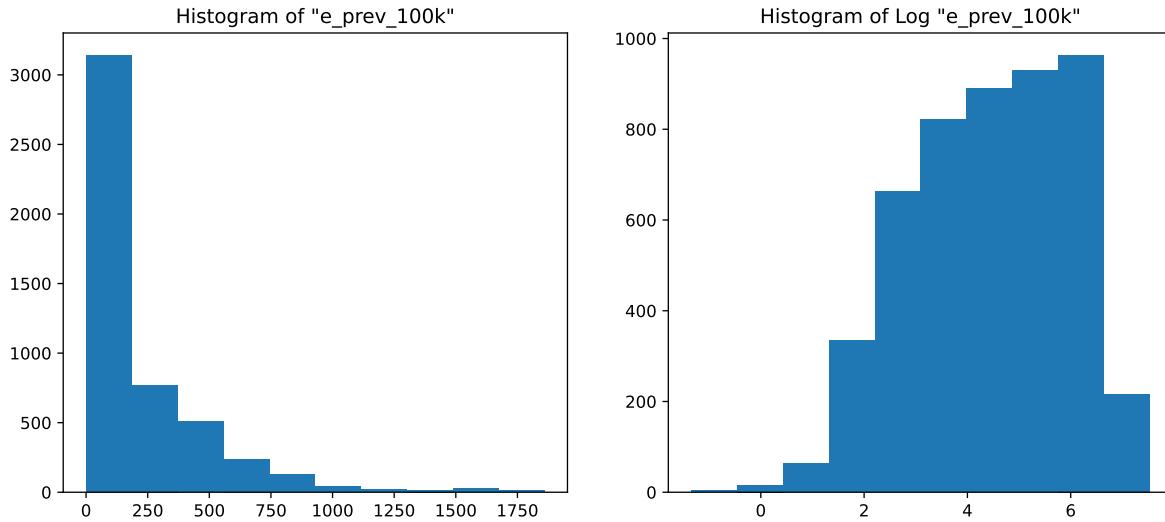
Go ahead and change the `colname` variable (question 1) into a different column name (that you can pick from the list you have just before question 1 (but do pick a numeric column)). And re-execute the code from question 1 and question 2 and you'll see your new Q-Q plot of the column you just picked.

3) Have a look at the slides from Week 03 for different shapes

Ok ? Now try to guess the shape of the distribution of our selected column (shape of its histogram) from its Q-Q Plot above.

4) Visualise the column on a histogram and reflect on whether the shape you inferred from Q-Q plots and the shape of the histogram correlate

```
# Histogramming the column we picked (not sure the verb exists though)
plt.figure(figsize=(12, 5))
plt.subplot(1,2,1)
plt.title('Histogram of "{}".format(colname)')
plt.hist(col)
plt.subplot(1,2,2)
plt.title('Histogram of Log "{}".format(colname)')
plt.hist(np.log(col));
```



Of course it does ! From the shape of the Q-Q Plot above (convex, slope upwards) and the Slide of Q-Q Plots from Week 3, we could conclude before looking at the histogram that our distribution was right tailed (or positively skewed if you're into complex vocabulary lol). And it is !

14.3 Part III : Distributions, Sampling

1) Inspecting the effect of sample size on descriptive statistics

```
# Defining a few variables so we can change their values easily without having to change
n = [5, 20, 100, 2000]
stds = [0.5, 1, 3]

# Initializing empty 2D arrays where we're going to store the results of our simulation
mean = np.empty([len(n), len(stds)])
std = np.empty([len(n), len(stds)])
skewness = np.empty([len(n), len(stds)])
kurtos = np.empty([len(n), len(stds)])

# Conducting the experiments and storing the results in the respective 2D arrays
for i, sample_size in enumerate(n):
    for j, theoretical_std in enumerate(stds):
        sample = np.random.normal(loc=0, scale=theoretical_std, size=sample_size)
        mean[i,j] = sample.mean()
```

```

        std[i,j] = sample.std()
        skewness[i,j] = skew(sample)
        kurtos[i,j] = kurtosis(sample)

# Turning the mean 2D array into a pandas dataframe
mean = pd.DataFrame(mean, columns = stds, index = n)
mean = mean.rename_axis('Sample Size').rename_axis("Standard Deviation", axis="columns")

# Turning the std 2D array into a pandas dataframe
std = pd.DataFrame(std, columns = stds, index = n)
std = std.rename_axis('Sample Size').rename_axis("Standard Deviation", axis="columns")

# Turning the skewness 2D array into a pandas dataframe
skewness = pd.DataFrame(skewness, columns = stds, index = n)
skewness = skewness.rename_axis('Sample Size').rename_axis("Standard Deviation", axis="columns")

# Turning the kurtosis 2D array into a pandas dataframe
kurtos = pd.DataFrame(kurtos, columns = stds, index = n)
kurtos = kurtos.rename_axis('Sample Size').rename_axis("Standard Deviation", axis="columns")

print("GAUSSIAN DISTRIBUTION\n")
print('Results for the Mean :')
mean # This is a dataframe containing the means of the samples generated with different va
print('Results for the Standard Deviation :')
std # This is a dataframe containing the standard deviations of the samples generated with
print('Results for the Skewness :')
skewness # This is a dataframe containing the skews of the samples generated with differen
print('Results for the Kurtosis :')
kurtos # This is a dataframe containing the kurtosis of the samples generated with differen

```

GAUSSIAN DISTRIBUTION

Results for the Mean :

Results for the Standard Deviation :

Results for the Skewness :

Results for the Kurtosis :

	Standard Deviation	0.5	1.0	3.0
	Sample Size			
5		-1.375086	-0.886408	-1.023777

Standard Deviation	0.5	1.0	3.0
Sample Size			
20	-0.175535	-0.932631	1.079546
100	0.128930	0.932878	-0.362034
2000	0.195084	-0.058148	-0.116846

Basically, the more data you have (the bigger your sample), the more accurate your empirical estimates are going to be. Observe for example the values of mean (1st DataFrame) and variance (2nd DataFrame) for the 2000 sample size (last row). In the first one, the values are all close to 0, because we generated our sample from a Gaussian with mean 0, and the values in the second one are all close to the values in the column names (which refer to the variance of the distribution of the sample). This means that for with a sample size of 2000, our estimates are really close to the “True” values (with which we generated the sample). Also, the Skew of a Gaussian distribution should be 0, and it is confirmed in the 3rd DataFrame where the values are close to 0 in the last row (i.e big sample size).

2) Same as before but with a Poisson distribution (which has just one parameter lambda instead of 2 like the gaussian)

```
# Defining a few variables se we can change their values easiliy without having to change
n = [5, 20, 100, 2000]
lambd = [0.5, 1, 3] # In a gaussian we had two parameters, a var specified here and a mean
#everywhere. Here we have one parameter called lambda.

# Initializing empty 2D arrays where we're going to store the results of our simulation
mean = np.empty([len(n), len(lambd)])
std = np.empty([len(n), len(lambd)])
skewness = np.empty([len(n), len(lambd)])
kurtos = np.empty([len(n), len(lambd)])

# Conducting the experiments and storing the results in the respective 2D arrays
for i, sample_size in enumerate(n):
    for j, theoritical_lambd in enumerate(lambd):
        ****************************
        sample = np.random.poisson(lam = theoritical_lambd, size = sample_size) # THIS IS
        ****************************
        mean[i,j] = sample.mean()
        std[i,j] = sample.std()
        skewness[i,j] = skew(sample)
        kurtos[i,j] = kurtosis(sample)
```

```

# Turning the mean 2D array into a pandas dataframe
mean = pd.DataFrame(mean, columns = lambd, index = n)
mean = mean.rename_axis('Sample Size').rename_axis("Lambda", axis="columns")

# Turning the std 2D array into a pandas dataframe
std = pd.DataFrame(std, columns = lambd, index = n)
std = std.rename_axis('Sample Size').rename_axis("Lambda", axis="columns")

# Turning the skewness 2D array into a pandas dataframe
skewness = pd.DataFrame(skewness, columns = lambd, index = n)
skewness = skewness.rename_axis('Sample Size').rename_axis("Lambda", axis="columns")

# Turning the kurtosis 2D array into a pandas dataframe
kurtos = pd.DataFrame(kurtos, columns = lambd, index = n)
kurtos = kurtos.rename_axis('Sample Size').rename_axis("Lambda", axis="columns")

print("POISSON DISTRIBUTION\n")
print('Results for the Mean :')
mean # This is a dataframe containing the means of the samples generated with different va
print('Results for the Standard Deviation :')
std # This is a dataframe containing the standard deviations of the samples generated with
#and sample size
print('Results for the Skewness :')
skewness # This is a dataframe containing the skews of the samples generated with differen
#size
print('Results for the Kurtosis :')
kurtos # This is a dataframe containing the kurtosis of the samples generated with differe
#size

```

POISSON DISTRIBUTION

```

Results for the Mean :
Results for the Standard Deviation :
Results for the Skewness :
Results for the Kurtosis :

```

Lambda	0.5	1.0	3.0
Sample Size			
5	NaN	-1.833333	-0.068998

Lambda	0.5	1.0	3.0
Sample Size			
20	-1.153061	-1.085465	0.689567
100	0.623251	0.136173	-0.303323
2000	2.183012	1.528798	0.469243

Just remember, the lambda parameter that defines the Poisson distribution is also the mean of the distribution. This is confirmed in the first DataFrame where the values (means of samples) are close to the column labels (theoretical lambda which is also equal to theoretical mean), especially in the last row.

14.4 Part IV : Robust Statistics

- 1) Choose a number of columns with different shapes, for instance, “e_prev_100k_hi” is left skewed or some columns where the variation is high or you notice potential outliers. You can make use of a series of boxplots to exploratively analyse the data for outliers

```
# Listing the columns
df_tuber.columns

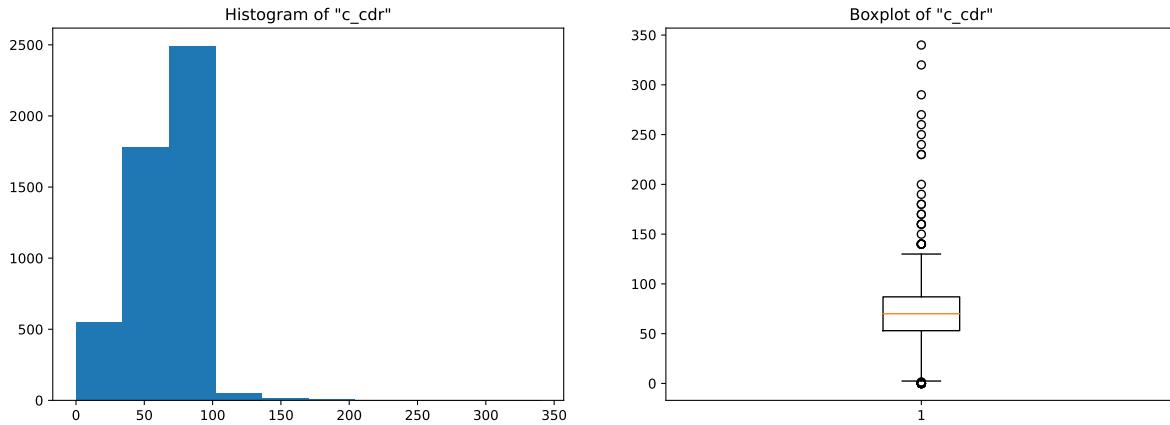
Index(['country', 'iso2', 'iso3', 'iso_numeric', 'g_whoregion', 'year',
       'e_pop_num', 'e_prev_100k', 'e_prev_100k_lo', 'e_prev_100k_hi',
       'e_prev_num', 'e_prev_num_lo', 'e_prev_num_hi', 'e_mort_exc_tbhiv_100k',
       'e_mort_exc_tbhiv_100k_lo', 'e_mort_exc_tbhiv_100k_hi',
       'e_mort_exc_tbhiv_num', 'e_mort_exc_tbhiv_num_lo',
       'e_mort_exc_tbhiv_num_hi', 'source_mort', 'e_inc_100k', 'e_inc_100k_lo',
       'e_inc_100k_hi', 'e_inc_num', 'e_inc_num_lo', 'e_inc_num_hi',
       'e_tbhiv_prct', 'e_tbhiv_prct_lo', 'e_tbhiv_prct_hi',
       'e_inc_tbhiv_100k', 'e_inc_tbhiv_100k_lo', 'e_inc_tbhiv_100k_hi',
       'e_inc_tbhiv_num', 'e_inc_tbhiv_num_lo', 'e_inc_tbhiv_num_hi',
       'source_tbhiv', 'c_cdr', 'c_cdr_lo', 'c_cdr_hi'],
      dtype='object')
```

```
# Alright I already know a few columns with outliers but let's try to find them together
colname = 'c_cdr' # change the column name by choosing different ones from above (numeric
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.hist(df_tuber[colname])
```

```

plt.title('Histogram of "{}".format(colname))
plt.subplot(1,2,2)
plt.boxplot(df_tuber[colname]);
plt.title('Boxplot of "{}".format(colname));

```



```

# Chosen columns : I picked 3, feel free to change them and experiment
chosen_colnames = ['e_pop_num', 'e_prev_100k', 'c_cdr']

```

- 2) For the chosen columns, estimate both the conventional and the robust descriptive statistics and compare. Observe how these pairs deviate from each other based on the characteristics of the underlying data

```

# Central Tendency : Mean vs Median (Median is the robust version of the mean, because it
# the ordering of the points and not the actual values like the mean does)
df_tuber[chosen_colnames].describe().loc[['mean', '50%'], : ] # The 50% is the median (50%

```

	e_pop_num	e_prev_100k	c_cdr
mean	2.899179e+07	207.694422	67.570706
50%	5.140332e+06	93.000000	70.000000

Look at how the values are different between the mean and the median ... LOOOOOOK ! This is why when you have a skewed (unsymmetrical) distribution it's usually more interesting to use the median as a measure of the central tendency of the data. One important thing to note here, for the two first attributes, the mean is higher than the median, but for the last it's the opposite. This can tell you a thing or two about the shape of your distribution : if the mean is higher than the median, this means that the distribution is skewed to the right (right tail) which pulls the mean higher. And vice-versa.

Moral of the story is ... outliers are a pain in the a**.

```
# Spread : Standard Deviation vs Inter-Quartile Range vs Median Absolute Deviation (MAD)
stds = df_tuber[chosen_colnames].std()
iqr = df_tuber[chosen_colnames].quantile(0.75) - df_tuber[chosen_colnames].quantile(0.25)
medianAD = mad(df_tuber[chosen_colnames])

output = pd.DataFrame(stds, columns = ['std']).T
output = pd.concat([output, pd.DataFrame(iqr, columns = ['IQR']).T], ignore_index=False)
output = pd.concat([output, pd.DataFrame(medianAD, columns = ['MAD']), index = chosen_colnames])
output
```

	e_pop_num	e_prev_100k	c_cdr
std	1.177827e+08	269.418159	25.234773
IQR	1.677193e+07	280.500000	34.000000
MAD	7.454908e+06	120.090780	25.204238

The values here are different as well, maybe more so for the “e_pop_num” attribute than the others, but that is just because of the scaling : “e_pop_num” takes big values overall compared to the other columns, which you can check with the mean values right above.

For the first attribute, the standard deviation is higher, and both the IQR and MAD are close to each other. For the second attribute, the inter-quartile range is slightly higher than the standard deviation, but the MAD is far below (less than half) the other two values, and the reason for that is a little bit involved : Basically, the standard deviation measures the spread by computing the squared deviation from the mean while the median absolute deviation evaluates the spread by computing the absolute deviation. This means that when the outliers have much bigger values than the “normal” points, the squared difference explodes (figuratively of course ;p) compared to the absolute difference. And this is actually the case for our second distribution (e_prev_100k) where most values are between 50 and 300 while many outliers lay above the 750 mark and go all the way up to 1800 (look at the boxplots below). For the third attribute the values are somewhat close, especially the std and the MAD, that’s because if you inspect the boxplot, this column doesn’t have many outliers to begin with.

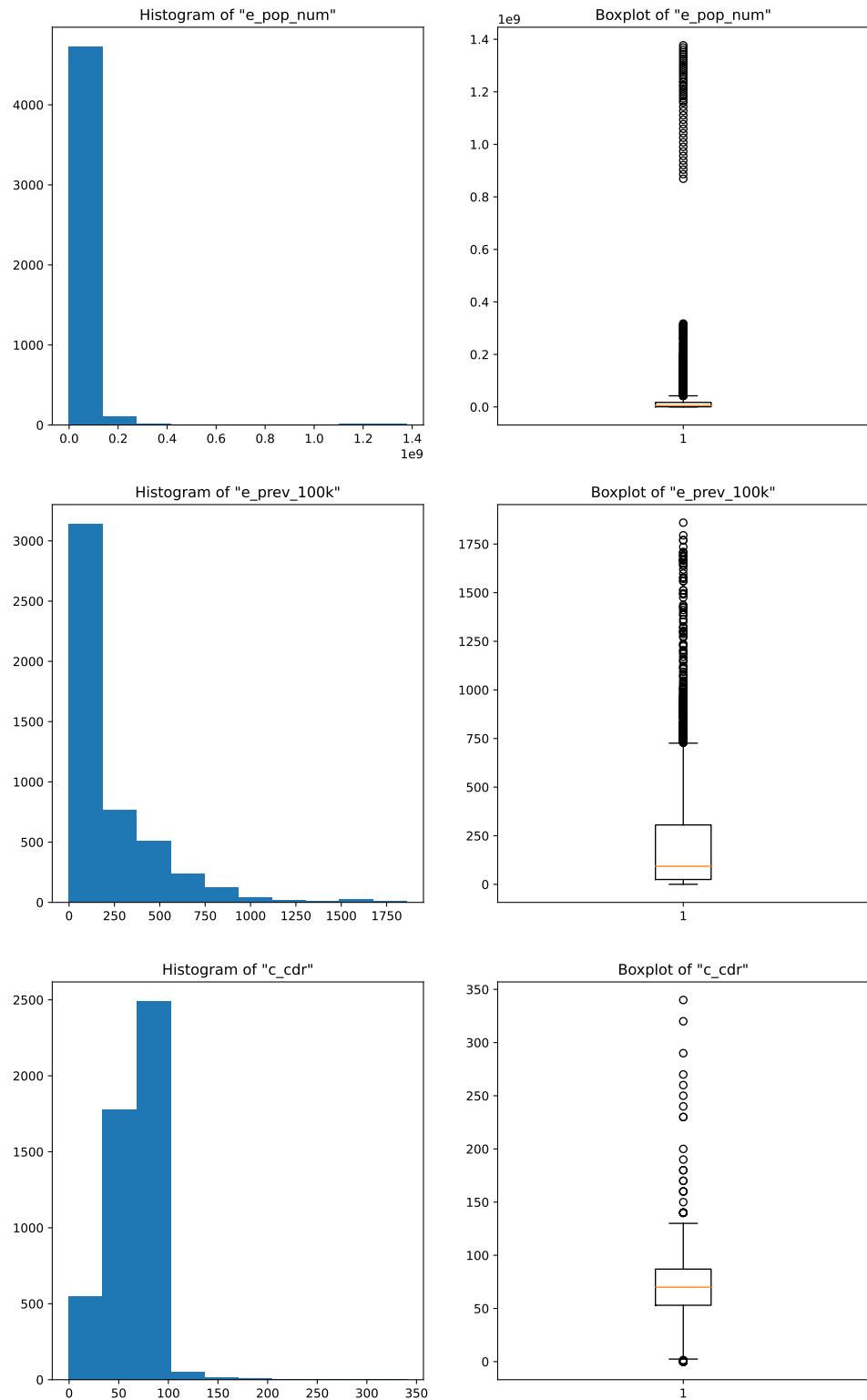
Nonetheless, the differences are real, and if we don’t want to have to handle outliers, then we should be using robust statistics like the median to describe the central tendency and inter-quartile range or median absolute deviation to measure the spread of our data.

```
# Boxplots of the different columns
plt.figure(figsize=(12,20))
```

```
plt.subplot(3,2,1)
plt.hist(df_tuber[chosen_colnames[0]])
plt.title('Histogram of "{}".format(chosen_colnames[0])))
plt.subplot(3,2,2)
plt.boxplot(df_tuber[chosen_colnames[0]])
plt.title('Boxplot of "{}".format(chosen_colnames[0])))

plt.subplot(3,2,3)
plt.hist(df_tuber[chosen_colnames[1]])
plt.title('Histogram of "{}".format(chosen_colnames[1])))
plt.subplot(3,2,4)
plt.boxplot(df_tuber[chosen_colnames[1]])
plt.title('Boxplot of "{}".format(chosen_colnames[1])))

plt.subplot(3,2,5)
plt.hist(df_tuber[chosen_colnames[2]])
plt.title('Histogram of "{}".format(chosen_colnames[2])))
plt.subplot(3,2,6)
plt.boxplot(df_tuber[chosen_colnames[2]])
plt.title('Boxplot of "{}".format(chosen_colnames[2]));
```



This is it. I hope this jupyter notebook finds you well.

May the Python divinities be with you !

15 IM939 - Lab 3 - Regression

Our aim is to introduce simple linear regression in Python.

15.1 Scikit Learn

We are going to use Scikit Learn library. More information about the librry you can find here: https://www.tutorialspoint.com/scikit_learn/scikit_learn_linear_regression.htm

15.2 Sea Ice Dataset

The script below is the based on the Sea Ice dataset (Igual, Segui 2017). You can find the data here: <ftp://sidads.colorado.edu/DATASETS/NOAA/> or read more about the project here: https://nsidc.org/data/seacie_index

This notebook will walk you through an example of a simple linear regression and the other notebook “IM939 Lab 3 - Linear Regression Exercise.ipynb” will include the example of a multiple linear regression, too.

15.3 Simple and Multiple Linear Regression

In the **linear model** the response \mathbf{y} depends linearly from the covariates \mathbf{x}_i .

In the **simple** linear regression, with a single variable, we described the relationship between the predictor and the response with a straight line. The general linear model:

$$\mathbf{y} = a_0 + a_1 \mathbf{x}_1$$

The parameter a_0 is called the *constant* term or the *intercept*.

In the case of **multiple** linear regression we extend this idea by fitting a m-dimensional hyperplane to our m predictors.

$$\mathbf{y} = a_1 \mathbf{x}_1 + \dots + a_m \mathbf{x}_m$$

The a_i are termed the *parameters* of the model or the coefficients.

15.4 Ordinary Least Squares

Ordinary Least Squares (OLS) is the simplest and most common **estimator** in which the coefficients a 's of the simple linear regression: $\mathbf{y} = a_0 + a_1 \mathbf{x}$, are chosen to minimize the **square of the distance between the predicted values and the actual values**.

$$\|a_0 + a_1 \mathbf{x} - \mathbf{y}\|_2^2 = \sum_{j=1}^n (a_0 + a_1 x_j - y_j)^2,$$

This expression is often called **sum of squared errors of prediction (SSE)**.

16 Case study: Climate Change and Sea Ice Extent

The question: Has there been a decrease in the amount of ice in the last years?

16.1 Reading Data

There are five steps. First, let's upload the data that is already in the folder: SeaIce.txt. It is a text file.

The file 'SeaIce.txt' is a Tab separated file containing: + Year: 4-digit year + mo: 1- or 2-digit month + data_type: Input data set (Goddard/NRTSI-G) + region: Hemisphere that this data covers (N: Northern; S: Southern) + extent: Sea ice extent in millions of square km + area: Sea ice area in millions of square km

Once we upload the data, we can create a *DataFrame* using Pandas. A reminder what is DataFrame: <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html>

```
import pandas as pd
ice = pd.read_csv('data/raw/SeaIce.txt',
                   delim_whitespace=True)

print('shape:', ice.shape) #this returns number of rows and columns in a dataset
ice.head()
```

shape: (424, 6)

	year	mo	data_type	region	extent	area
0	1979	1	Goddard	N	15.54	12.33
1	1980	1	Goddard	N	14.96	11.85
2	1981	1	Goddard	N	15.03	11.82
3	1982	1	Goddard	N	15.26	12.11
4	1983	1	Goddard	N	15.10	11.92

We can compute the mean for that interval of time (1981 through 2010), before data cleaning.

```
ice.mean(numeric_only = True)

year      1996.000000
mo         6.500000
extent    -35.443066
area      -37.921108
dtype: float64
```

Did we receive a negative mean...?

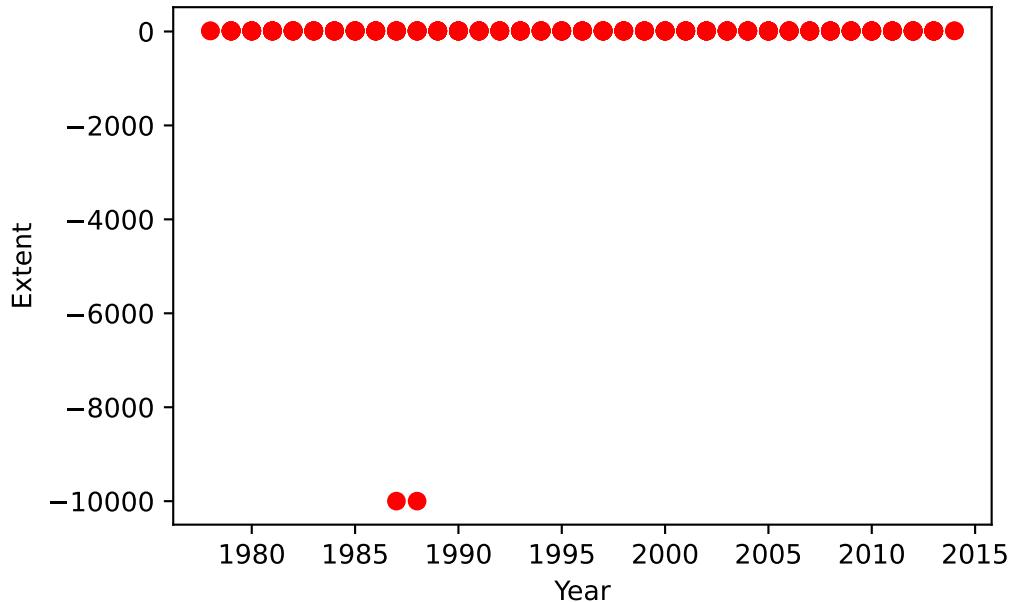
16.2 Data visualisation to explore data

Do you remember Seaborn? We will use lmplot() function from Seaborn to explore linear relationship of different forms, e.g. relationship between the month of the year (variable) and extent (responses):

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

# Visualize the data
x = ice.year
y = ice.extent
plt.scatter(x, y, color = 'red')
plt.xlabel('Year')
plt.ylabel('Extent')

Text(0, 0.5, 'Extent')
```



We detect some outlier or missing data. we are going to use function np.unique and find the unique elements of an array.

```
?np.unique

print ('Different values in data_type field:', np.unique(ice.data_type.values)) # there

Different values in data_type field: ['-9999' 'Goddard' 'NRTSI-G']
```

Let's see what type of data we have, other than the ones printed above

```
print (ice[(ice.data_type != 'Goddard') & (ice.data_type != 'NRTSI-G')])
```

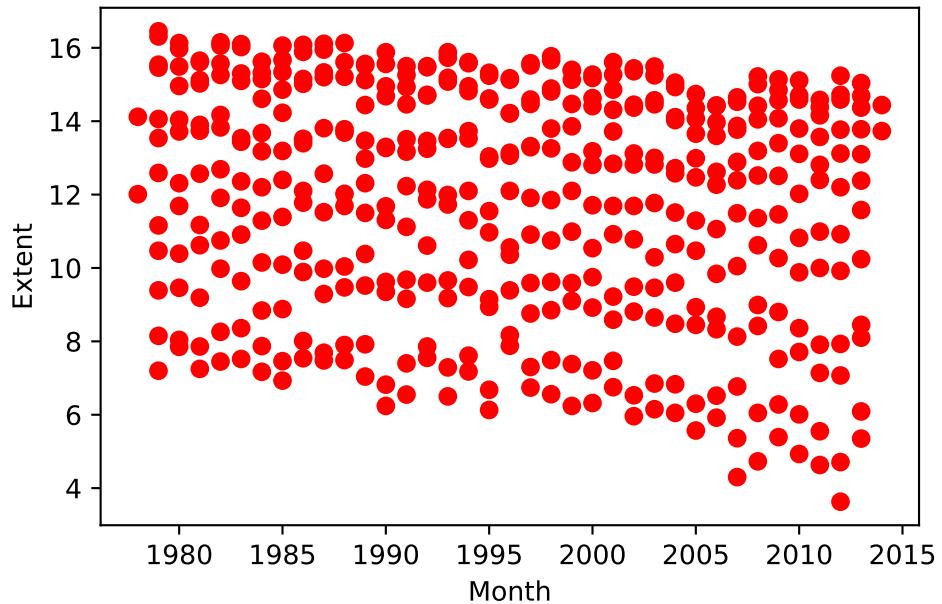
	year	mo	data_type	region	extent	area
9	1988	1	-9999	N	-9999.0	-9999.0
397	1987	12	-9999	N	-9999.0	-9999.0

Data cleaning: we checked all the values and notice -9999 values in data_type field which should contain 'Goddard' or 'NRTSI-G' (some type of input dataset). So we clean them by removing these instances

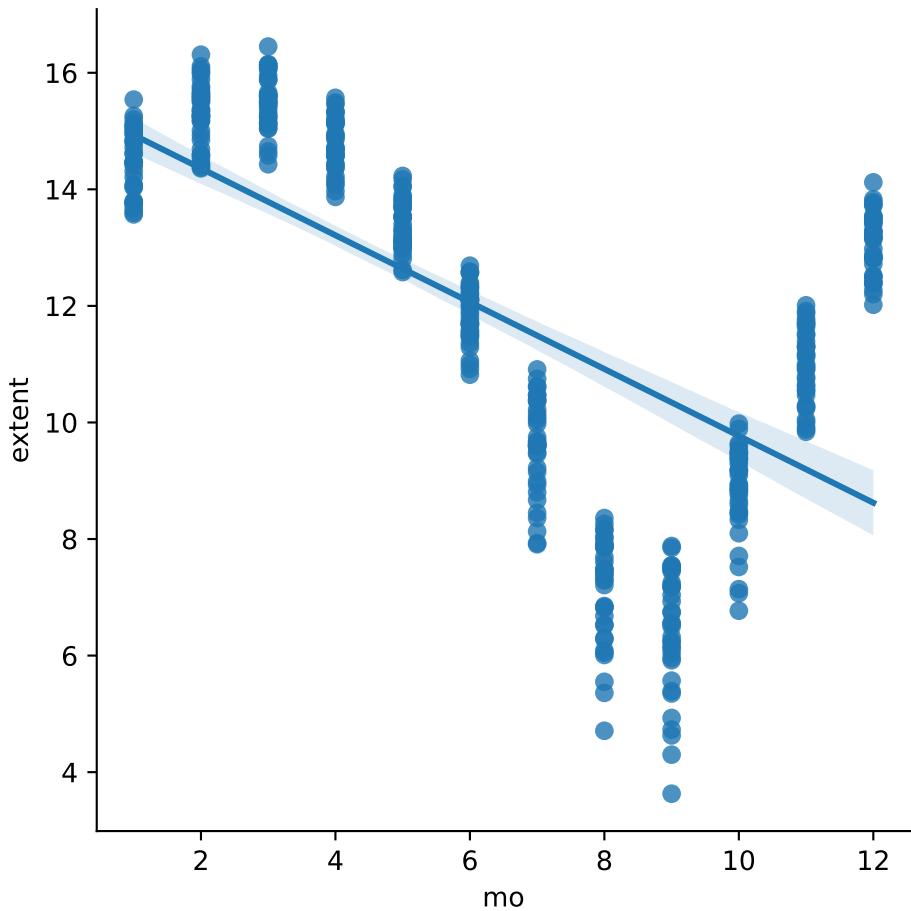
```
# We can easily clean the data now:  
ice2 = ice[ice.data_type != '-9999']  
print ('shape:', ice2.shape)  
# And repeat the plot  
x = ice2.year  
y = ice2.extent  
plt.scatter(x, y, color = 'red')  
plt.xlabel('Month')  
plt.ylabel('Extent')
```

shape: (422, 6)

Text(0, 0.5, 'Extent')



```
sns.lmplot(data = ice2, x = "mo", y = "extent")
```

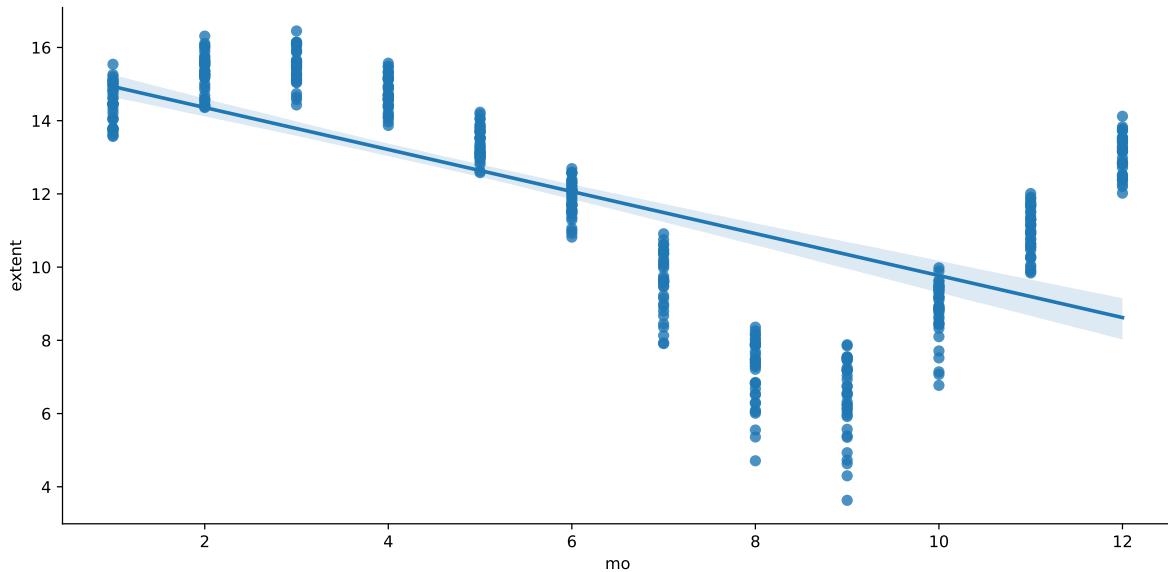


Above you can see ice extent data by month. You can see a monthly fluctuation of the sea ice extent, as would be expected for the different seasons of the year. In order to run regression, and avoid this fluctuation we can normalize data. This will let us see the evolution of the extent over the years.

16.3 Normalization

The `lmplot()` function from the Seaborn module is intended for exploring linear relationships of different forms in multidimensional datasets. Input data must be in a Pandas DataFrame. To plot them, we provide the predictor and response variable names along with the dataset

```
sns.lmplot(ice2, x = "mo", y = "extent", height = 5.2, aspect = 2);
plt.savefig("IceExtentCleanedByMonth.png", dpi = 300, bbox_inches = 'tight')
```



```
# Compute the mean for each month.
grouped = ice2.groupby('mo')
month_means = grouped.extent.mean()
month_variances = grouped.extent.var()
print ('Means:', month_means)
print ('Variances:', month_variances)
```

```
Means: mo
1      14.479429
2      15.298889
3      15.491714
4      14.766000
5      13.396000
6      11.860000
7      9.601143
8      7.122286
9      6.404857
10     8.809143
11     10.964722
12     13.059429
Name: extent, dtype: float64
Variances: mo
1      0.304906
2      0.295804
```

```

3    0.237209
4    0.215378
5    0.189901
6    0.247918
7    0.679175
8    0.824577
9    1.143902
10   0.630361
11   0.412511
12   0.284870
Name: extent, dtype: float64

```

To capture variation per month, we can compute mean for the i-th interval of time (using 1979-2014) and subtract it from the set of extent values for that month . This can be converted to a relative percentage difference by dividing it by the total average (1979-2014) and multiplying by 100.

```

# Data normalization
for i in range(12):
    ice2.extent[ice2.mo == i+1] = 100*(ice2.extent[ice2.mo == i+1] - month_means[i+1])/month_mean

sns.lmplot(ice2 , x = "mo", y = "extent", height = 5.2, aspect = 2);
plt.savefig("IceExtentNormalizedByMonth.png", dpi = 300, bbox_inches = 'tight')

```

```
/var/folders/7v/zl9mv52s3ls94kntlt_19ryh0000gq/T/ipykernel_65977/3597698861.py:3: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#inplace-operations
`ice2.extent[ice2.mo == i+1] = 100*(ice2.extent[ice2.mo == i+1] - month_means[i+1])/month_mean`

```
/var/folders/7v/zl9mv52s3ls94kntlt_19ryh0000gq/T/ipykernel_65977/3597698861.py:3: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#inplace-operations
`ice2.extent[ice2.mo == i+1] = 100*(ice2.extent[ice2.mo == i+1] - month_means[i+1])/month_mean`

```
/var/folders/7v/zl9mv52s3ls94kntlt_19ryh0000gq/T/ipykernel_65977/3597698861.py:3: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#inplace-operations
`ice2.extent[ice2.mo == i+1] = 100*(ice2.extent[ice2.mo == i+1] - month_means[i+1])/month_mean`

```
/var/folders/7v/zl9mv52s3ls94kntlt_19ryh0000gq/T/ipykernel_65977/3597698861.py:3: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#inplace-operations
ice2.extent[ice2.mo == i+1] = 100*(ice2.extent[ice2.mo == i+1] - month_means[i+1])/month_mean
/var/folders/7v/zl9mv52s3ls94kntlt_19ryh0000gq/T/ipykernel_65977/3597698861.py:3: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#inplace-operations
ice2.extent[ice2.mo == i+1] = 100*(ice2.extent[ice2.mo == i+1] - month_means[i+1])/month_mean
/var/folders/7v/zl9mv52s3ls94kntlt_19ryh0000gq/T/ipykernel_65977/3597698861.py:3: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#inplace-operations
ice2.extent[ice2.mo == i+1] = 100*(ice2.extent[ice2.mo == i+1] - month_means[i+1])/month_mean
/var/folders/7v/zl9mv52s3ls94kntlt_19ryh0000gq/T/ipykernel_65977/3597698861.py:3: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#inplace-operations
ice2.extent[ice2.mo == i+1] = 100*(ice2.extent[ice2.mo == i+1] - month_means[i+1])/month_mean
/var/folders/7v/zl9mv52s3ls94kntlt_19ryh0000gq/T/ipykernel_65977/3597698861.py:3: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

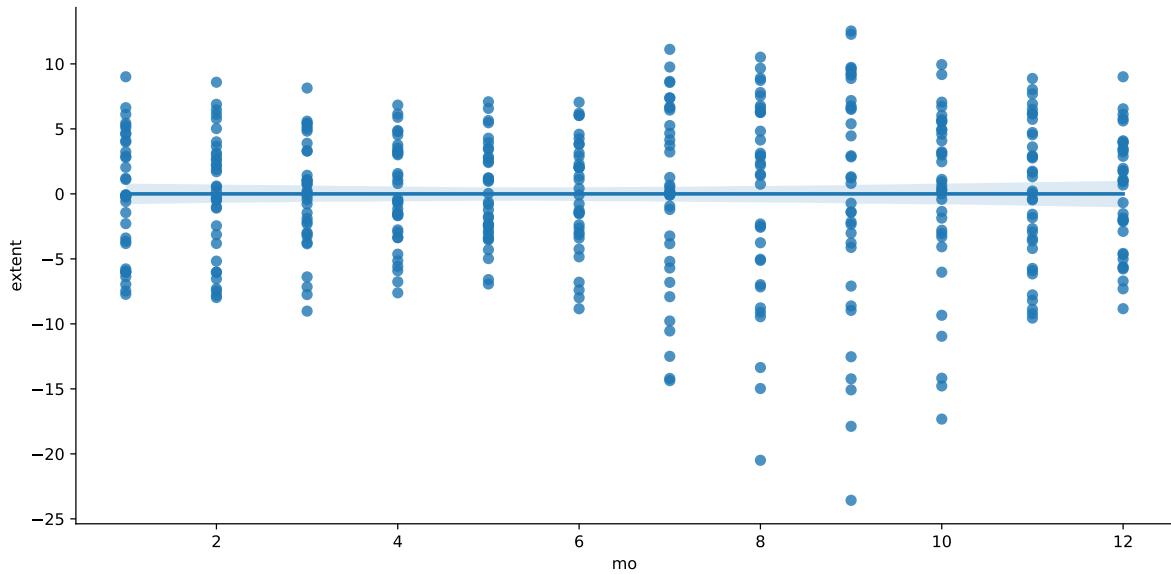
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#inplace-operations
ice2.extent[ice2.mo == i+1] = 100*(ice2.extent[ice2.mo == i+1] - month_means[i+1])/month_mean
/var/folders/7v/zl9mv52s3ls94kntlt_19ryh0000gq/T/ipykernel_65977/3597698861.py:3: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#inplace-operations
ice2.extent[ice2.mo == i+1] = 100*(ice2.extent[ice2.mo == i+1] - month_means[i+1])/month_mean
/var/folders/7v/zl9mv52s3ls94kntlt_19ryh0000gq/T/ipykernel_65977/3597698861.py:3: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#inplace-operations
ice2.extent[ice2.mo == i+1] = 100*(ice2.extent[ice2.mo == i+1] - month_means[i+1])/month_mean
/var/folders/7v/zl9mv52s3ls94kntlt_19ryh0000gq/T/ipykernel_65977/3597698861.py:3: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#inplace-operations
ice2.extent[ice2.mo == i+1] = 100*(ice2.extent[ice2.mo == i+1] - month_means[i+1])/month_mean
/var/folders/7v/zl9mv52s3ls94kntlt_19ryh0000gq/T/ipykernel_65977/3597698861.py:3: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#inplace-operations
ice2.extent[ice2.mo == i+1] = 100*(ice2.extent[ice2.mo == i+1] - month_means[i+1])/month_mean

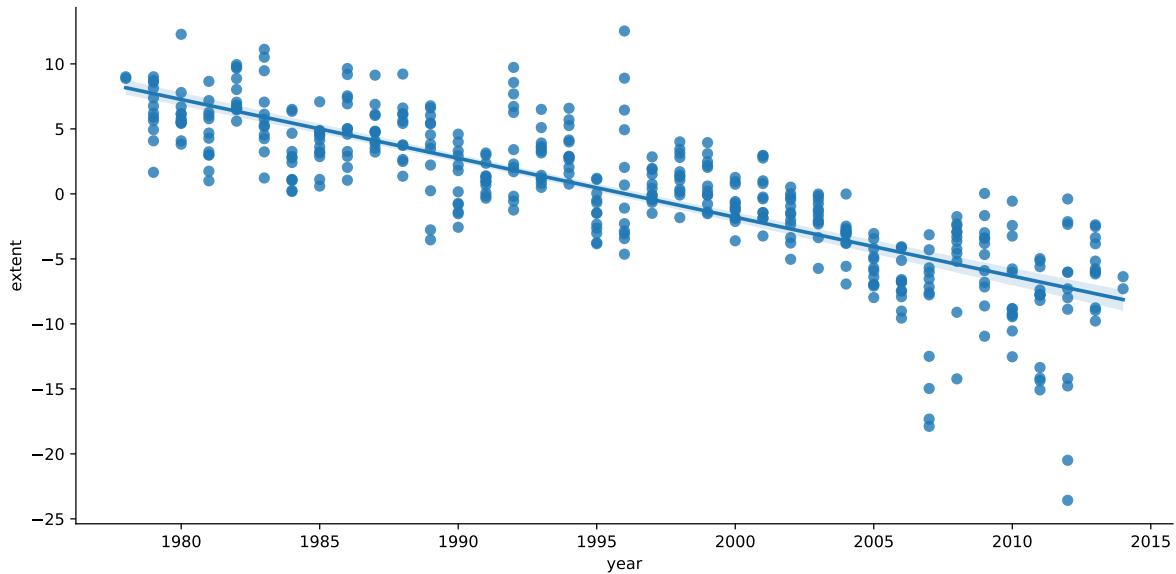


```
print ('mean:', ice2.extent.mean())
print ('var:', ice2.extent.var())
```

```
mean: -7.745252569896827e-16
var: 31.983239774968798
```

Let us consider the entire year

```
sns.lmplot(ice2, x = "year", y = "extent", height = 5.2, aspect = 2);
plt.savefig("IceExtentAllMonthsByYearlmplot.png", dpi = 300, bbox_inches = 'tight')
```



16.4 Pearson's correlation

Let's calculate Pearson's correlation coefficient and the p-value for testing non-correlation.

```
import scipy.stats
scipy.stats.pearsonr(ice2.year.values, ice2.extent.values)

PearsonRResult(statistic=-0.8183500709897178, pvalue=4.4492318168687107e-103)
```

16.5 Simple OLS

We can also compute the trend as a simple linear regression (OLS) and quantitatively evaluate it.

For that we use using **Scikit-learn**, library that provides a variety of both supervised and unsupervised machine learning techniques. Scikit-learn provides an object-oriented interface centered around the concept of an Estimator. The Estimator.fit method sets the state of the estimator based on the training data. Usually, the data is comprised of a two-dimensional numpy array X of shape (n_samples, n_predictors) that holds the so-called feature matrix and a one-dimensional numpy array y that holds the responses. Some estimators allow the user to control the fitting behavior. For example, the `sklearn.linear_model.LinearRegression` estimator allows the user to specify whether or not to fit an intercept term. This is done by

setting the corresponding constructor arguments of the estimator object. During the fitting process, the state of the estimator is stored in instance attributes that have a trailing underscore (''). *For example, the coefficients of a LinearRegression estimator are stored in the attribute `coef`.*

Estimators that can generate predictions provide a `Estimator.predict` method. In the case of regression, `Estimator.predict` will return the predicted regression values, \hat{y} .

```
from sklearn.linear_model import LinearRegression

est = LinearRegression(fit_intercept = True)

x = ice2[['year']]
y = ice2[['extent']]

est.fit(x, y)

print("Coefficients:", est.coef_)
print ("Intercept:", est.intercept_)
```

```
Coefficients: [[-0.45275459]]
Intercept: [903.71640207]
```

We can evaluate the model fitting by computing the mean squared error (MSE) and the coefficient of determination (R^2) of the model. The coefficient R^2 is defined as $(1 - \mathbf{u}/\mathbf{v})$, where \mathbf{u} is the residual sum of squares $\sum(y - \hat{y})^2$ and \mathbf{v} is the regression sum of squares $\sum(y - \bar{y})^2$, where \bar{y} is the mean. The best possible score for R^2 is 1.0: lower values are worse. These measures can provide a quantitative answer to the question we are facing: Is there a negative trend in the evolution of sea ice extent over recent years?

```
from sklearn import metrics

# Analysis for all months together.
x = ice2[['year']]
y = ice2[['extent']]
model = LinearRegression()
model.fit(x, y)
y_hat = model.predict(x)
plt.plot(x, y, 'o', alpha = 0.5)
plt.plot(x, y_hat, 'r', alpha = 0.5)
plt.xlabel('year')
plt.ylabel('extent (All months)')
```

```

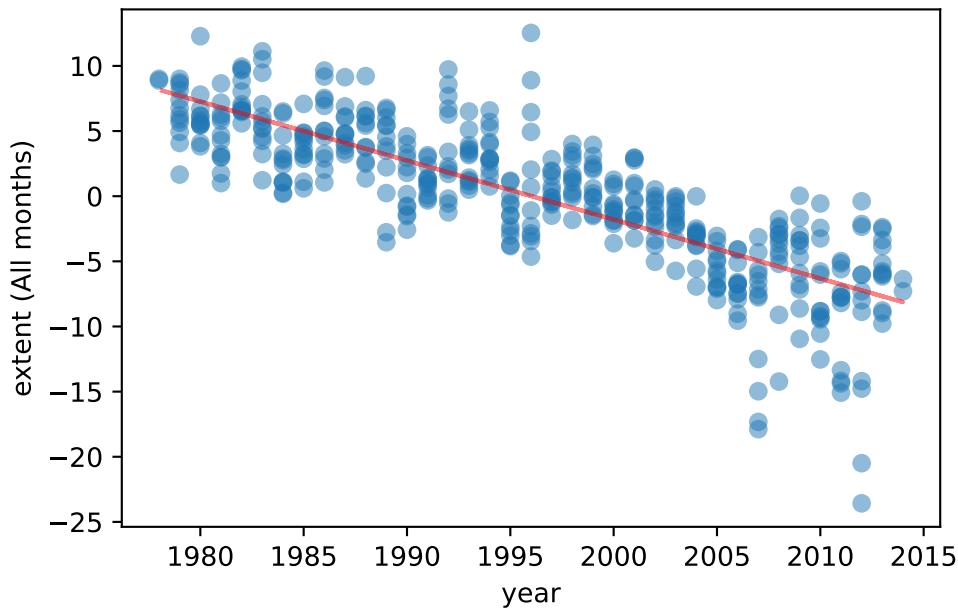
print ("MSE:", metrics.mean_squared_error(y_hat, y))
print ("R^2:", metrics.r2_score(y_hat, y))
print ("var:", y.var())
plt.savefig("IceExtentLinearRegressionAllMonthsByYearPrediction.png", dpi = 300, bbox_inches="tight")

```

```

MSE: 10.539131639803488
R^2: 0.5067870382100226
var: extent      31.98324
dtype: float64

```



We can conclude that the data show a long-term negative trend in recent years.

```

# Analysis for a particular month.
#For January
jan = ice2[ice2.mo == 1];

x = jan[['year']]
y = jan[['extent']]

model = LinearRegression()
model.fit(x, y)

```

```

y_hat = model.predict(x)

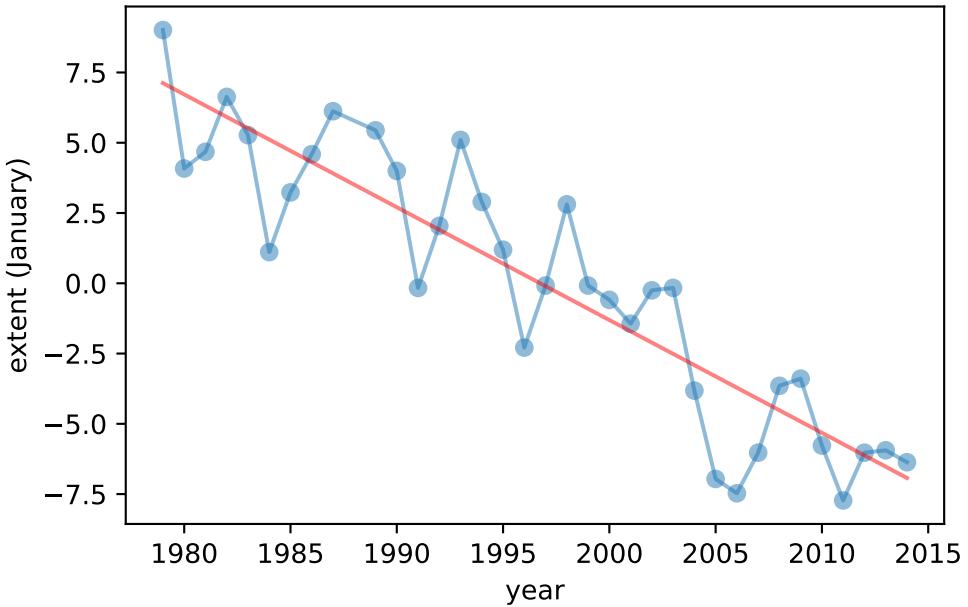
plt.figure()
plt.plot(x, y, '-o', alpha = 0.5)
plt.plot(x, y_hat, 'r', alpha = 0.5)
plt.xlabel('year')
plt.ylabel('extent (January)')

print ("MSE:", metrics.mean_squared_error(y_hat, y))
print ("R^2:", metrics.r2_score(y_hat, y))

```

MSE: 3.8395160752867565

R²: 0.7810636041396216



We can also estimate the extent value for 2025. For that we use the function predict of the model.

```

X = np.array(2025)
y_hat = model.predict(X.reshape(-1, 1))
j = 1 # January
# Original value (before normalization)
y_hat = (y_hat*month_means.mean()/100) + month_means[j]

```

```
print ("Prediction of extent for January 2025 (in millions of square km):", y_hat)
```

Prediction of extent for January 2025 (in millions of square km): [[13.14449923]]

```
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/sklearn/base.py:464: UserWarning:  
    warnings.warn(
```

Prediction of extent for January 2025 (in millions of square km): [13.14449923]

17 IM939 - Lab 3 - Regression Exercise

Now it's your turn to prepare a linear regression model.

17.1 Scikit Learn

You can use here as well Scikit Learn library. More information you can find here: https://www.tutorialspoint.com/scikit_learn/scikit_learn_linear_regression.htm

17.2 Wine Dataset

More information about the dataset here: <https://archive.ics.uci.edu/ml/datasets/wine+quality>
What would be your research question? What do you like to learn, given the data you have?

17.3 Reading Data

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

wine = pd.read_excel('data/raw/winequality-red_v2.xlsx', engine = 'openpyxl')

#You might need to use encoding, then the code will look like:
# wine = pd.read_excel('data/raw/winequality-red_v2.xlsx', engine = 'openpyxl', encoding='
```

17.4 Data exploration

Let's check the data, their distribution and central tendencies

```
print('shape:', wine.shape)
wine.head()
```

shape: (1599, 12)

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sul
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0

17.4.1 Check your variables

Use lmplot() function from Seaborn to explore linear relationship Input data must be in a Pandas DataFrame. To plot them, we provide the predictor and response variable names along with the dataset

Did you find outliers or missing data? You can use function np.unique and find the unique elements of an array.

```
?np.unique
```

Do you need to remove any cases?

Did you need to standarize data?

If you standarized data, try to plot them again

You can calculates a Pearson correlation coefficient and the p-value for testing non-correlation.

```
import scipy.stats
scipy.stats.pearsonr(wine.???.values, wine.???.values)
```

SyntaxError: invalid syntax (987973612.py, line 2)

using **Scikit-learn**, build a simple linear regression (OLS)

```

from sklearn.linear_model import LinearRegression

est = LinearRegression(fit_intercept = True)

x = wine[['???'}}
y = wine[['???'}

est.fit(x, y)

print("Coefficients:", est.coef_)
print ("Intercept:", est.intercept_)

```

KeyError: "None of [Index(['???'}, dtype='object')] are in the [columns]"

What is the model's mean squared error (MSE) and the coefficient of determination (R^2) ?

```

from sklearn import metrics

# Analysis for all months together.
x = wdi[['???'}}
y = wdi[['???'}}
model = LinearRegression()
model.fit(x, y)
y_hat = model.predict(x)
plt.plot(x, y, 'o', alpha = 0.5)
plt.plot(x, y_hat, 'r', alpha = 0.5)
plt.xlabel('?')
plt.ylabel('?')
print ("MSE:", metrics.mean_squared_error(y_hat, y))
print ("R^2:", metrics.r2_score(y_hat, y))
print ("var:", y.var())
plt.savefig("?.png", dpi = 300, bbox_inches = 'tight')

```

NameError: name 'wdi' is not defined

What's the conclusion?

Part V

Structures & Spaces

18 IM939 Lab 4 - Part 1 - Iris

18.1 Data

Many datasets have a high number of dimensions. We are going to explore dimension reduction (principle component analysis) and clustering techniques.

The simple Iris dataset is great for introducing these methods.

```
from sklearn.datasets import load_iris

iris = load_iris()
```

The iris dataset is in an odd format.

```
type(iris)

sklearn.utils._bunch.Bunch
```

Following [this stackoverflow answer](#) we can convert it into the pandas dataframe format we know and love.

```
import pandas as pd

iris_df = pd.DataFrame(iris.data, columns = iris.feature_names)
iris_df.head()
```

	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

We will scale the data between 0 and 1 to be on the safe side. All we are doing is placing the data on the same scale which is often called Normalisation (see [this blog entry](#)).

Standardisation often means centering the data around the mean.

Some algorithms are sensitive to the size of variables. For example, if the sepal widths were in meters and the other variables in cm then an algorithm may underweight sepal widths. Normalising the data puts all the data on a single scale.

If you cannot choose between them then try it both ways. You could compare the result with your raw data, the normalised data and the standardised data.

```
from sklearn.preprocessing import MinMaxScaler
col_names = iris_df.columns
iris_df = pd.DataFrame(MinMaxScaler().fit_transform(iris_df))
iris_df.columns = col_names # Column names were lost, so we need to re-introduce
iris_df
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	0.222222	0.625000	0.067797	0.041667
1	0.166667	0.416667	0.067797	0.041667
2	0.111111	0.500000	0.050847	0.041667
3	0.083333	0.458333	0.084746	0.041667
4	0.194444	0.666667	0.067797	0.041667
...
145	0.666667	0.416667	0.711864	0.916667
146	0.555556	0.208333	0.677966	0.750000
147	0.611111	0.416667	0.711864	0.791667
148	0.527778	0.583333	0.745763	0.916667
149	0.444444	0.416667	0.694915	0.708333

```
iris_df.shape
```

```
(150, 4)
```

Great.

Our dataset shows us the length and width of both the sepal (leaf) and petals of 150 plants. The dataset is quite famous and you can find a [wikipedia page](#) with details of the dataset.

18.2 Questions

To motivate our exploration of the data, consider the sorts of questions we can ask:

- Are all our plants from the same species?
- Do some plants have similar leaf and petal sizes?
- Can we differentiate between the plants using all 4 variables (dimensions)?
- Do we need to include both length and width, or can we reduce these dimensions and simplify our analysis?

18.3 Initial exploration

We can explore a dataset with few variables using plots.

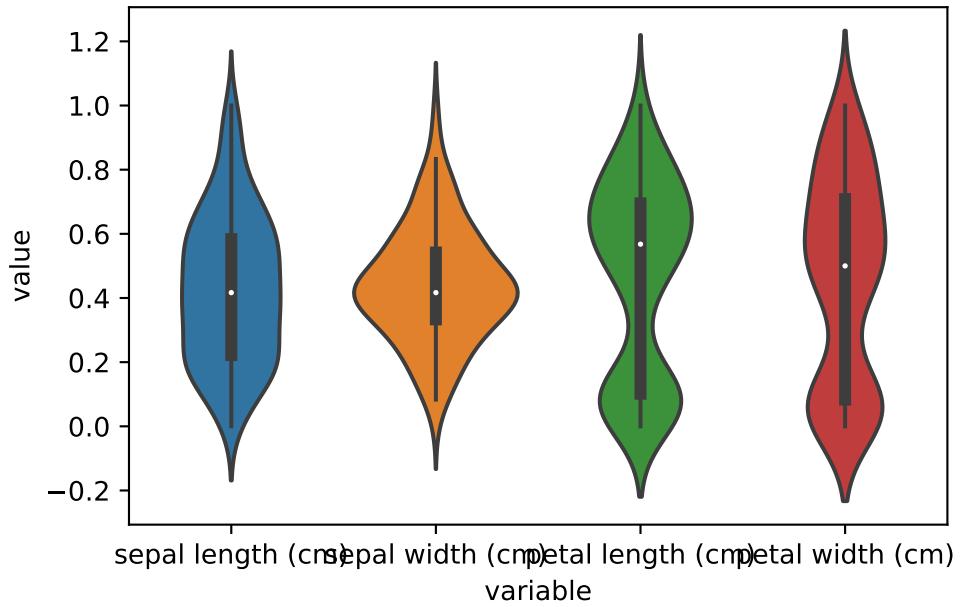
```
import seaborn as sns

# some plots require a long datafram structure
iris_df_long = iris_df.melt()
iris_df_long
```

	variable	value
0	sepal length (cm)	0.222222
1	sepal length (cm)	0.166667
2	sepal length (cm)	0.111111
3	sepal length (cm)	0.083333
4	sepal length (cm)	0.194444
...
595	petal width (cm)	0.916667
596	petal width (cm)	0.750000
597	petal width (cm)	0.791667
598	petal width (cm)	0.916667
599	petal width (cm)	0.708333

```
sns.violinplot(data = iris_df_long, x = 'variable', y = 'value')
```

```
<Axes: xlabel='variable', ylabel='value'>
```



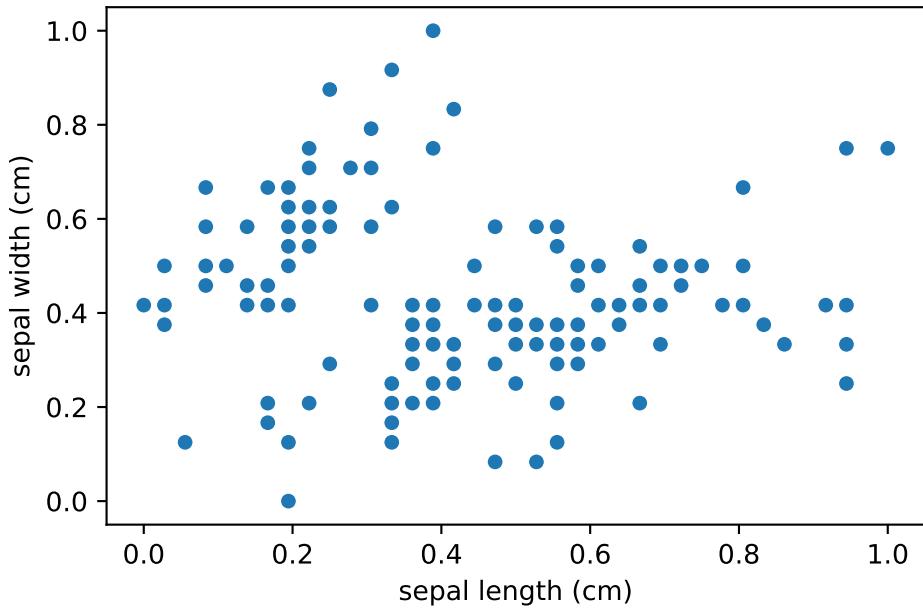
The below plots use the wide data structure.

```
iris_df
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	0.222222	0.625000	0.067797	0.041667
1	0.166667	0.416667	0.067797	0.041667
2	0.111111	0.500000	0.050847	0.041667
3	0.083333	0.458333	0.084746	0.041667
4	0.194444	0.666667	0.067797	0.041667
...
145	0.666667	0.416667	0.711864	0.916667
146	0.555556	0.208333	0.677966	0.750000
147	0.611111	0.416667	0.711864	0.791667
148	0.527778	0.583333	0.745763	0.916667
149	0.444444	0.416667	0.694915	0.708333

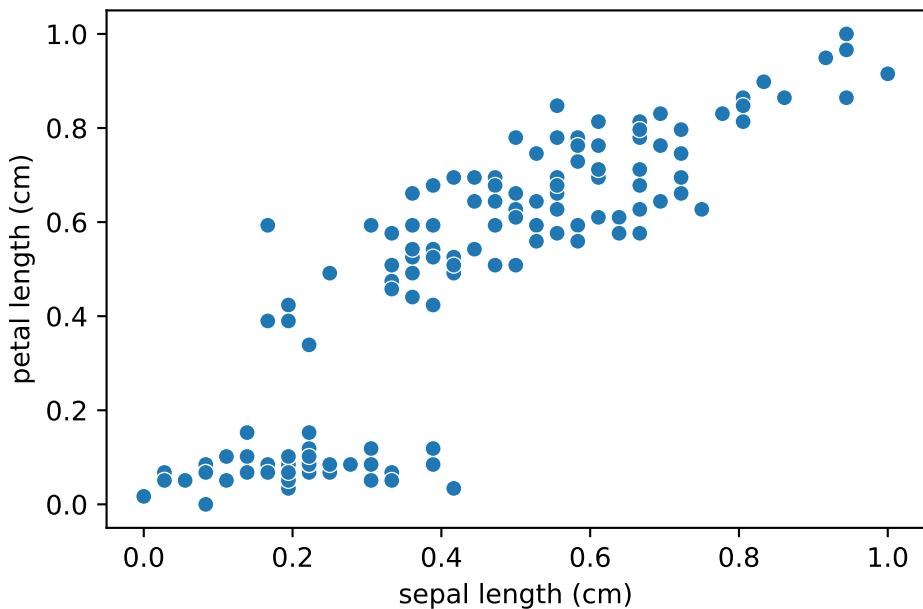
```
sns.scatterplot(data = iris_df, x = 'sepal length (cm)', y = 'sepal width (cm)')
```

```
<Axes: xlabel='sepal length (cm)', ylabel='sepal width (cm)'>
```



```
sns.scatterplot(data = iris_df, x = 'sepal length (cm)', y = 'petal length (cm)')
```

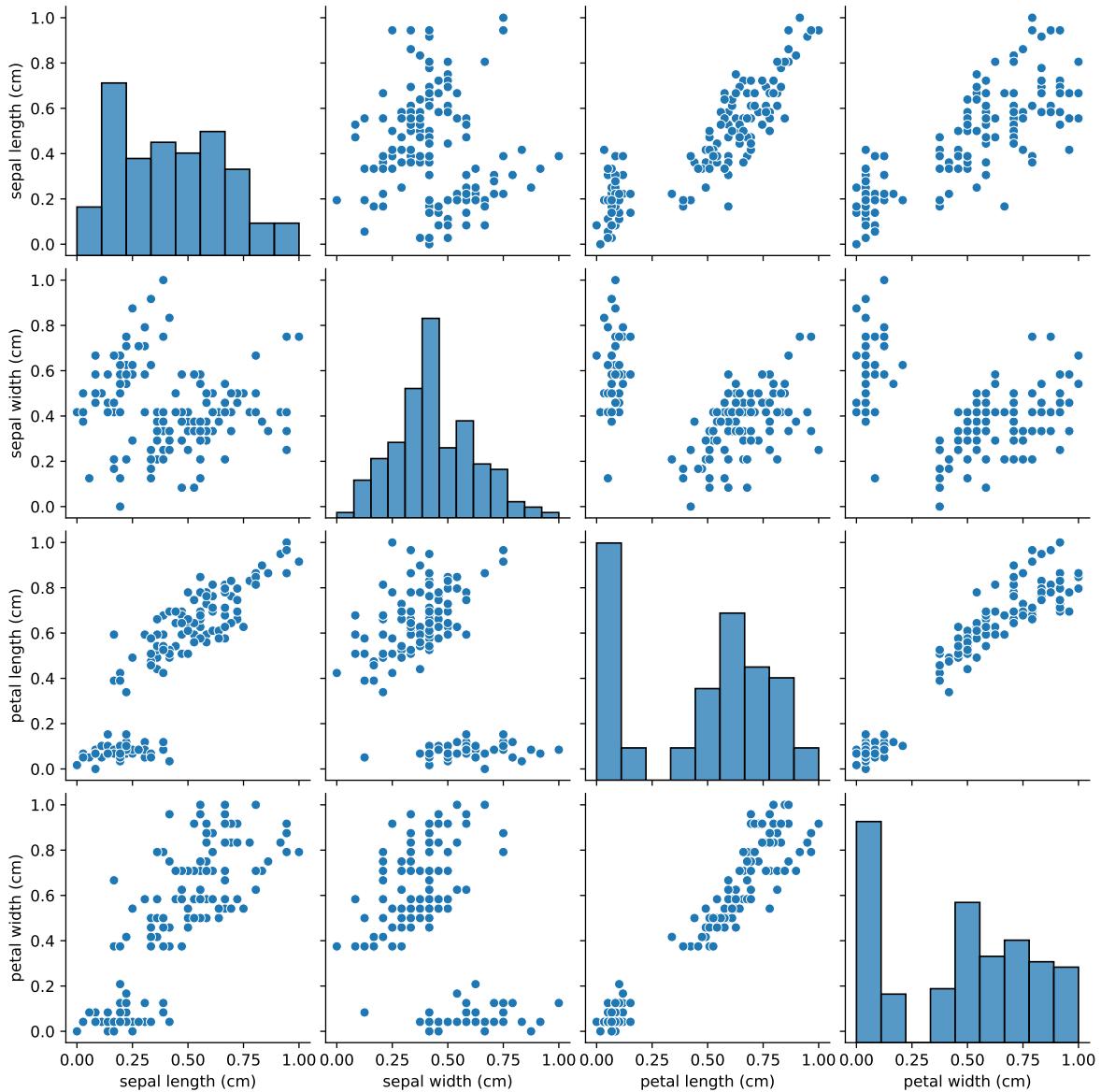
```
<Axes: xlabel='sepal length (cm)', ylabel='petal length (cm)'>
```



Interesting. There seem to be two groupings in the data.

It might be easier to look at all the variables at once.

```
sns.pairplot(iris_df)
```



There seem to be some groupings in the data. Though we cannot easily identify which point corresponds to which row.

18.4 Clustering

A cluster is simply a group based on similarity. There are several methods and we will use a relatively simple one called K-means clustering.

In K-means clustering an algorithm tries to group our items (plants in the iris dataset) based on similarity. We decide how many groups we want and the algorithm does the best it can (an accessible introduction to k-means clustering is [here](#)).

To start, we import the KMeans function from sklearn cluster module and turn our data into a matrix.

```
from sklearn.cluster import KMeans

iris = iris_df.values
iris

array([[0.22222222, 0.625      , 0.06779661, 0.04166667],
       [0.16666667, 0.41666667, 0.06779661, 0.04166667],
       [0.11111111, 0.5      , 0.05084746, 0.04166667],
       [0.08333333, 0.45833333, 0.08474576, 0.04166667],
       [0.19444444, 0.66666667, 0.06779661, 0.04166667],
       [0.30555556, 0.79166667, 0.11864407, 0.125      ],
       [0.08333333, 0.58333333, 0.06779661, 0.08333333],
       [0.19444444, 0.58333333, 0.08474576, 0.04166667],
       [0.02777778, 0.375      , 0.06779661, 0.04166667],
       [0.16666667, 0.45833333, 0.08474576, 0.      ],
       [0.30555556, 0.70833333, 0.08474576, 0.04166667],
       [0.13888889, 0.58333333, 0.10169492, 0.04166667],
       [0.13888889, 0.41666667, 0.06779661, 0.      ],
       [0.      , 0.41666667, 0.01694915, 0.      ],
       [0.41666667, 0.83333333, 0.03389831, 0.04166667],
       [0.38888889, 1.      , 0.08474576, 0.125      ],
       [0.30555556, 0.79166667, 0.05084746, 0.125      ],
       [0.22222222, 0.625      , 0.06779661, 0.08333333],
       [0.38888889, 0.75      , 0.11864407, 0.08333333],
       [0.22222222, 0.75      , 0.08474576, 0.08333333],
       [0.30555556, 0.58333333, 0.11864407, 0.04166667],
       [0.22222222, 0.70833333, 0.08474576, 0.125      ],
       [0.08333333, 0.66666667, 0.      , 0.04166667],
       [0.22222222, 0.54166667, 0.11864407, 0.16666667],
       [0.13888889, 0.58333333, 0.15254237, 0.04166667],
       [0.19444444, 0.41666667, 0.10169492, 0.04166667],
```

```

[0.19444444, 0.58333333, 0.10169492, 0.125      ],
[0.25      , 0.625      , 0.08474576, 0.04166667],
[0.25      , 0.58333333, 0.06779661, 0.04166667],
[0.11111111, 0.5      , 0.10169492, 0.04166667],
[0.13888889, 0.45833333, 0.10169492, 0.04166667],
[0.30555556, 0.58333333, 0.08474576, 0.125      ],
[0.25      , 0.875      , 0.08474576, 0.      ],
[0.33333333, 0.91666667, 0.06779661, 0.04166667],
[0.16666667, 0.45833333, 0.08474576, 0.04166667],
[0.19444444, 0.5      , 0.03389831, 0.04166667],
[0.33333333, 0.625      , 0.05084746, 0.04166667],
[0.16666667, 0.66666667, 0.06779661, 0.      ],
[0.02777778, 0.41666667, 0.05084746, 0.04166667],
[0.22222222, 0.58333333, 0.08474576, 0.04166667],
[0.19444444, 0.625      , 0.05084746, 0.08333333],
[0.05555556, 0.125      , 0.05084746, 0.08333333],
[0.02777778, 0.5      , 0.05084746, 0.04166667],
[0.19444444, 0.625      , 0.10169492, 0.20833333],
[0.22222222, 0.75      , 0.15254237, 0.125      ],
[0.13888889, 0.41666667, 0.06779661, 0.08333333],
[0.22222222, 0.75      , 0.10169492, 0.04166667],
[0.08333333, 0.5      , 0.06779661, 0.04166667],
[0.27777778, 0.70833333, 0.08474576, 0.04166667],
[0.19444444, 0.54166667, 0.06779661, 0.04166667],
[0.75      , 0.5      , 0.62711864, 0.54166667],
[0.58333333, 0.5      , 0.59322034, 0.58333333],
[0.72222222, 0.45833333, 0.66101695, 0.58333333],
[0.33333333, 0.125      , 0.50847458, 0.5      ],
[0.61111111, 0.33333333, 0.61016949, 0.58333333],
[0.38888889, 0.33333333, 0.59322034, 0.5      ],
[0.55555556, 0.54166667, 0.62711864, 0.625      ],
[0.16666667, 0.16666667, 0.38983051, 0.375      ],
[0.63888889, 0.375      , 0.61016949, 0.5      ],
[0.25      , 0.29166667, 0.49152542, 0.54166667],
[0.19444444, 0.      , 0.42372881, 0.375      ],
[0.44444444, 0.41666667, 0.54237288, 0.58333333],
[0.47222222, 0.08333333, 0.50847458, 0.375      ],
[0.5      , 0.375      , 0.62711864, 0.54166667],
[0.36111111, 0.375      , 0.44067797, 0.5      ],
[0.66666667, 0.45833333, 0.57627119, 0.54166667],
[0.36111111, 0.41666667, 0.59322034, 0.58333333],
[0.41666667, 0.29166667, 0.52542373, 0.375      ],
[0.52777778, 0.08333333, 0.59322034, 0.58333333],

```

[0.36111111, 0.20833333, 0.49152542, 0.41666667],
 [0.44444444, 0.5, 0.6440678, 0.70833333],
 [0.5, 0.33333333, 0.50847458, 0.5],
 [0.55555556, 0.20833333, 0.66101695, 0.58333333],
 [0.5, 0.33333333, 0.62711864, 0.45833333],
 [0.58333333, 0.375, 0.55932203, 0.5],
 [0.63888889, 0.41666667, 0.57627119, 0.54166667],
 [0.69444444, 0.33333333, 0.6440678, 0.54166667],
 [0.66666667, 0.41666667, 0.6779661, 0.66666667],
 [0.47222222, 0.375, 0.59322034, 0.58333333],
 [0.38888889, 0.25, 0.42372881, 0.375],
 [0.33333333, 0.16666667, 0.47457627, 0.41666667],
 [0.33333333, 0.16666667, 0.45762712, 0.375],
 [0.41666667, 0.29166667, 0.49152542, 0.45833333],
 [0.47222222, 0.29166667, 0.69491525, 0.625],
 [0.30555556, 0.41666667, 0.59322034, 0.58333333],
 [0.47222222, 0.58333333, 0.59322034, 0.625],
 [0.66666667, 0.45833333, 0.62711864, 0.58333333],
 [0.55555556, 0.125, 0.57627119, 0.5],
 [0.36111111, 0.41666667, 0.52542373, 0.5],
 [0.33333333, 0.20833333, 0.50847458, 0.5],
 [0.33333333, 0.25, 0.57627119, 0.45833333],
 [0.5, 0.41666667, 0.61016949, 0.54166667],
 [0.41666667, 0.25, 0.50847458, 0.45833333],
 [0.19444444, 0.125, 0.38983051, 0.375],
 [0.36111111, 0.29166667, 0.54237288, 0.5],
 [0.38888889, 0.41666667, 0.54237288, 0.45833333],
 [0.38888889, 0.375, 0.54237288, 0.5],
 [0.52777778, 0.375, 0.55932203, 0.5],
 [0.22222222, 0.20833333, 0.33898305, 0.41666667],
 [0.38888889, 0.33333333, 0.52542373, 0.5],
 [0.55555556, 0.54166667, 0.84745763, 1.],
 [0.41666667, 0.29166667, 0.69491525, 0.75],
 [0.77777778, 0.41666667, 0.83050847, 0.83333333],
 [0.55555556, 0.375, 0.77966102, 0.70833333],
 [0.61111111, 0.41666667, 0.81355932, 0.875],
 [0.91666667, 0.41666667, 0.94915254, 0.83333333],
 [0.16666667, 0.20833333, 0.59322034, 0.66666667],
 [0.83333333, 0.375, 0.89830508, 0.70833333],
 [0.66666667, 0.20833333, 0.81355932, 0.70833333],
 [0.80555556, 0.66666667, 0.86440678, 1.],
 [0.61111111, 0.5, 0.69491525, 0.79166667],
 [0.58333333, 0.29166667, 0.72881356, 0.75]

```
[0.69444444, 0.41666667, 0.76271186, 0.83333333],
[0.38888889, 0.20833333, 0.6779661 , 0.79166667],
[0.41666667, 0.33333333, 0.69491525, 0.95833333],
[0.58333333, 0.5      , 0.72881356, 0.91666667],
[0.61111111, 0.41666667, 0.76271186, 0.70833333],
[0.94444444, 0.75      , 0.96610169, 0.875      ],
[0.94444444, 0.25      , 1.      , 0.91666667],
[0.47222222, 0.08333333, 0.6779661 , 0.58333333],
[0.72222222, 0.5      , 0.79661017, 0.91666667],
[0.36111111, 0.33333333, 0.66101695, 0.79166667],
[0.94444444, 0.33333333, 0.96610169, 0.79166667],
[0.55555556, 0.29166667, 0.66101695, 0.70833333],
[0.66666667, 0.54166667, 0.79661017, 0.83333333],
[0.80555556, 0.5      , 0.84745763, 0.70833333],
[0.52777778, 0.33333333, 0.6440678 , 0.70833333],
[0.5      , 0.41666667, 0.66101695, 0.70833333],
[0.58333333, 0.33333333, 0.77966102, 0.83333333],
[0.80555556, 0.41666667, 0.81355932, 0.625      ],
[0.86111111, 0.33333333, 0.86440678, 0.75      ],
[1.      , 0.75      , 0.91525424, 0.79166667],
[0.58333333, 0.33333333, 0.77966102, 0.875      ],
[0.55555556, 0.33333333, 0.69491525, 0.58333333],
[0.5      , 0.25      , 0.77966102, 0.54166667],
[0.94444444, 0.41666667, 0.86440678, 0.91666667],
[0.55555556, 0.58333333, 0.77966102, 0.95833333],
[0.58333333, 0.45833333, 0.76271186, 0.70833333],
[0.47222222, 0.41666667, 0.6440678 , 0.70833333],
[0.72222222, 0.45833333, 0.74576271, 0.83333333],
[0.66666667, 0.45833333, 0.77966102, 0.95833333],
[0.72222222, 0.45833333, 0.69491525, 0.91666667],
[0.41666667, 0.29166667, 0.69491525, 0.75      ],
[0.69444444, 0.5      , 0.83050847, 0.91666667],
[0.66666667, 0.54166667, 0.79661017, 1.      ],
[0.66666667, 0.41666667, 0.71186441, 0.91666667],
[0.55555556, 0.20833333, 0.6779661 , 0.75      ],
[0.61111111, 0.41666667, 0.71186441, 0.79166667],
[0.52777778, 0.58333333, 0.74576271, 0.91666667],
[0.44444444, 0.41666667, 0.69491525, 0.70833333])
```

Specify our number of clusters.

```
k_means = KMeans(n_clusters = 3, init = 'random', n_init = 10)
```

Fit our kmeans model to the data

```
k_means.fit(iris)
```

```
KMeans(init='random', n_clusters=3, n_init=10)
```

The algorithm has assigned the a label to each row.

```
k_means.labels_
```

Each row has been assigned a label.

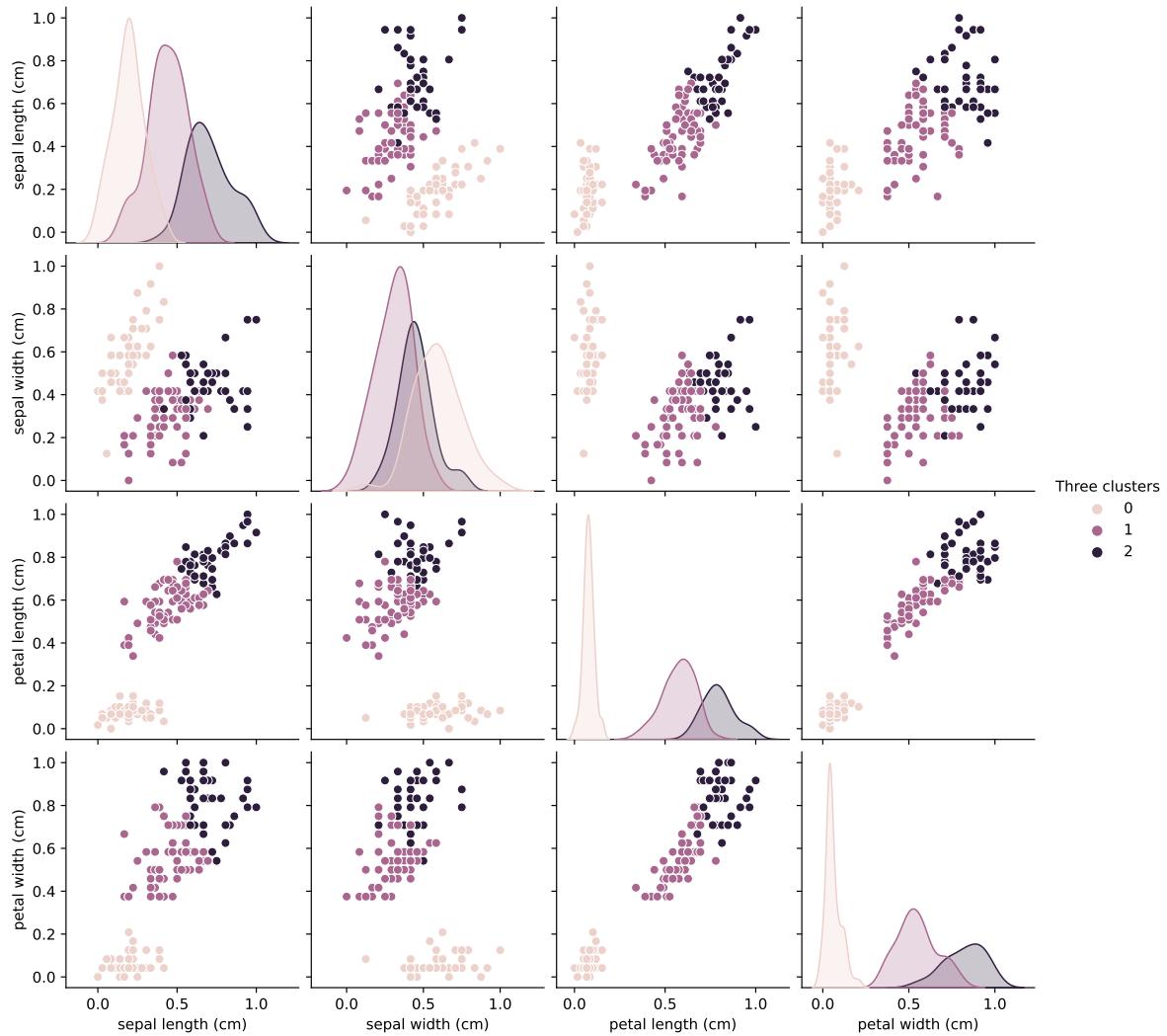
To tidy things up we should put everything into a dataframe.

```
iris_df['Three clusters'] = pd.Series(kmeans.predict(iris_df.values), index = iris_df.index)
```

iris_df

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Three clusters
0	0.222222	0.625000	0.067797	0.041667	0
1	0.166667	0.416667	0.067797	0.041667	0
2	0.111111	0.500000	0.050847	0.041667	0
3	0.083333	0.458333	0.084746	0.041667	0
4	0.194444	0.666667	0.067797	0.041667	0
...
145	0.666667	0.416667	0.711864	0.916667	2
146	0.555556	0.208333	0.677966	0.750000	1
147	0.611111	0.416667	0.711864	0.791667	2
148	0.527778	0.583333	0.745763	0.916667	2
149	0.444444	0.416667	0.694915	0.708333	1

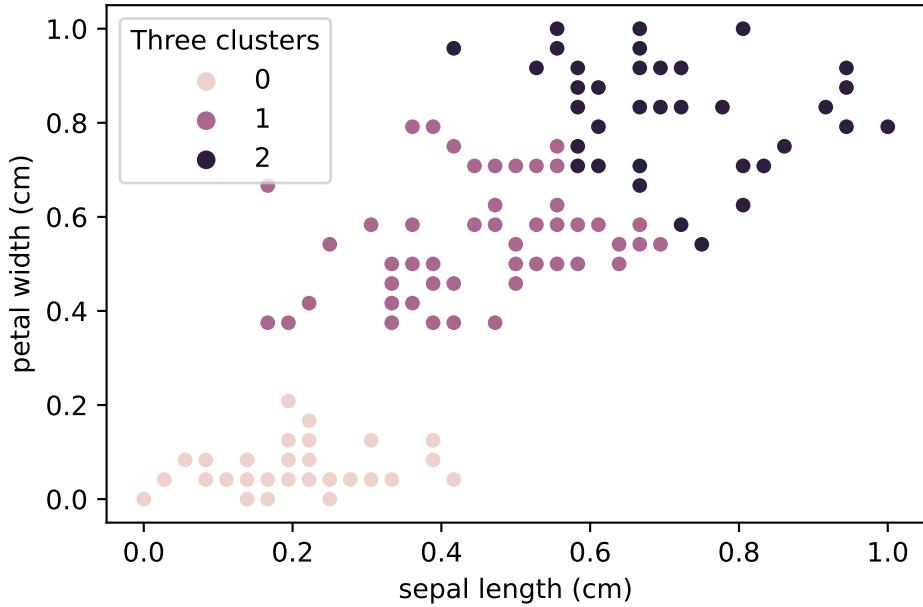
```
sns.pairplot(iris_df, hue = 'Three clusters')
```



That seems quite nice. We can also do individual plots if preferred.

```
sns.scatterplot(data = iris_df, x = 'sepal length (cm)', y = 'petal width (cm)', hue = 'Three clusters')
```

```
<Axes: xlabel='sepal length (cm)', ylabel='petal width (cm)'>
```



K-means works by clustering the data around central points (often called centroids, means or cluster centers). We can extract the cluster centres from the `kmeans` object.

```
k_means.cluster_centers_
```

```
array([[0.19611111, 0.595      , 0.07830508, 0.06083333],
       [0.44125683, 0.30737705, 0.57571548, 0.54918033],
       [0.70726496, 0.4508547 , 0.79704476, 0.82478632]])
```

It is tricky to plot these using seaborn but we can use a normal matplotlib scatter plot.

Let us grab the groups.

```
group1 = iris_df[iris_df['Three clusters'] == 0]
group2 = iris_df[iris_df['Three clusters'] == 1]
group3 = iris_df[iris_df['Three clusters'] == 2]
```

Grab the centroids

```
import pandas as pd

centres = k_means.cluster_centers_
```

```
data = {'x': [centres[0][0], centres[1][0], centres[2][0]],
        'y': [centres[0][3], centres[1][3], centres[2][3]]}
```

```
df = pd.DataFrame (data, columns = ['x', 'y'])
```

Create the plot

```
import matplotlib.pyplot as plt

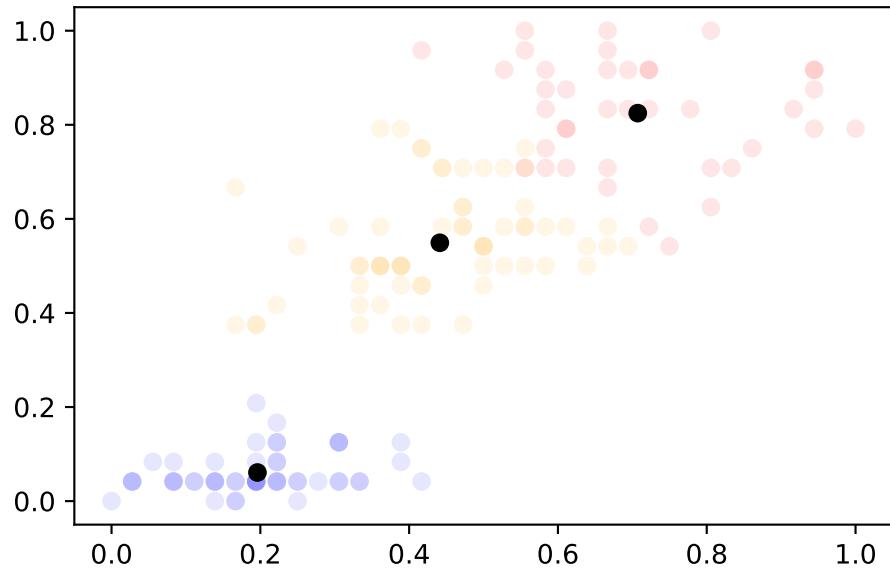
# Plot each group individually
plt.scatter(
    x = group1['sepal length (cm)'],
    y = group1['petal width (cm)'],
    alpha = 0.1, color = 'blue'
)

plt.scatter(
    x = group2['sepal length (cm)'],
    y = group2['petal width (cm)'],
    alpha = 0.1, color = 'orange'
)

plt.scatter(
    x = group3['sepal length (cm)'],
    y = group3['petal width (cm)'],
    alpha = 0.1, color = 'red'
)

# Plot cluster centres
plt.scatter(
    x = df['x'],
    y = df['y'],
    alpha = 1, color = 'black'
)
```

<matplotlib.collections.PathCollection at 0x13f2493d0>



18.5 Number of clusters

What happens if we change the number of clusters?

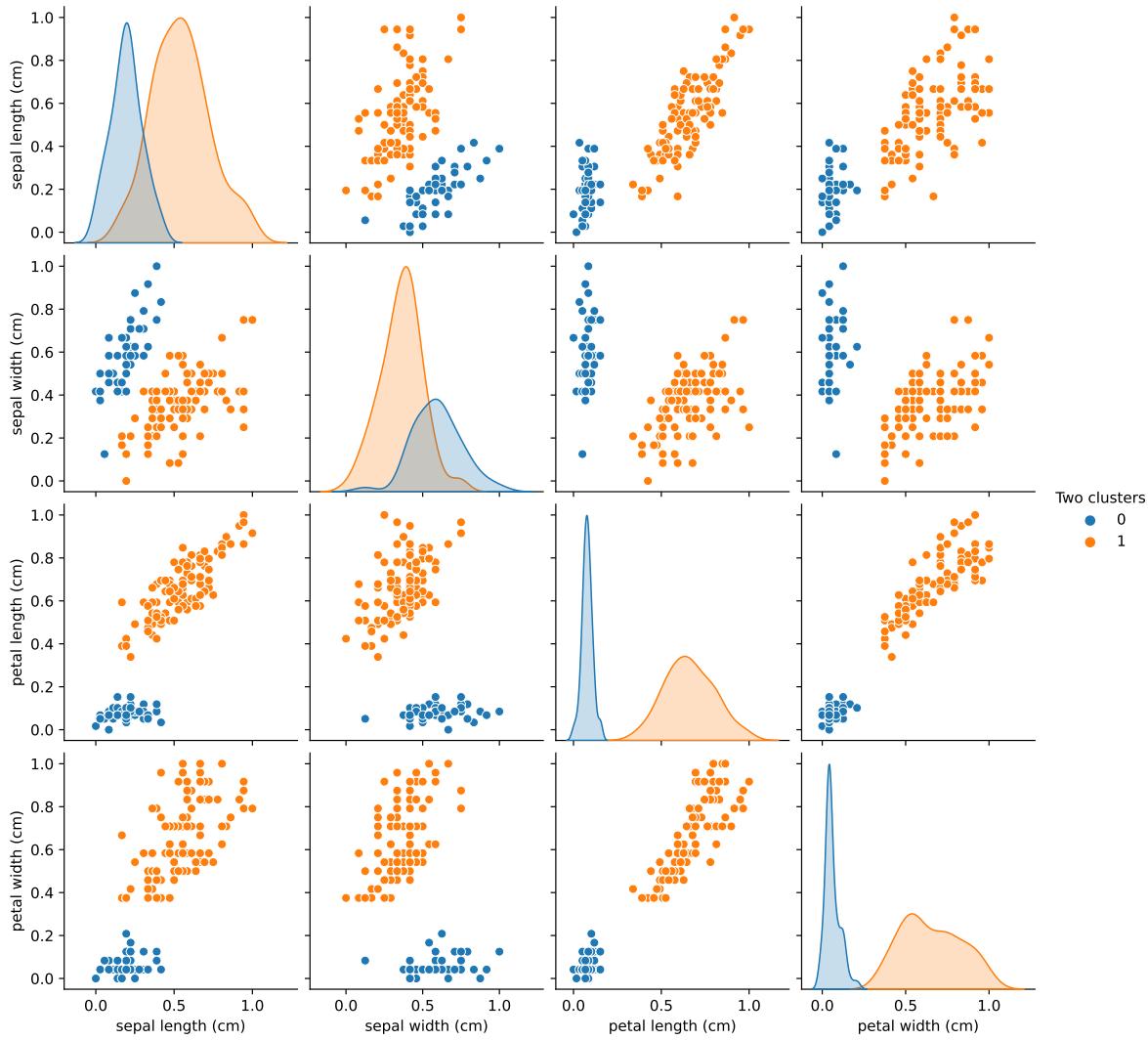
Two groups

```
k_means_2 = KMeans(n_clusters = 2, init = 'random', n_init = 10)
k_means_2.fit(iris)
iris_df['Two clusters'] = pd.Series(k_means_2.predict(iris_df.iloc[:,0:4].values), index =
```

Note that I have added a new column to the iris dataframe called ‘cluster 2 means’ and pass only our original 4 columns to the predict function (hence me using .iloc[:,0:4]).

How do our groupings look now (without plotting the cluster column)?

```
sns.pairplot(iris_df.loc[:, iris_df.columns != 'Three clusters'], hue = 'Two clusters')
```

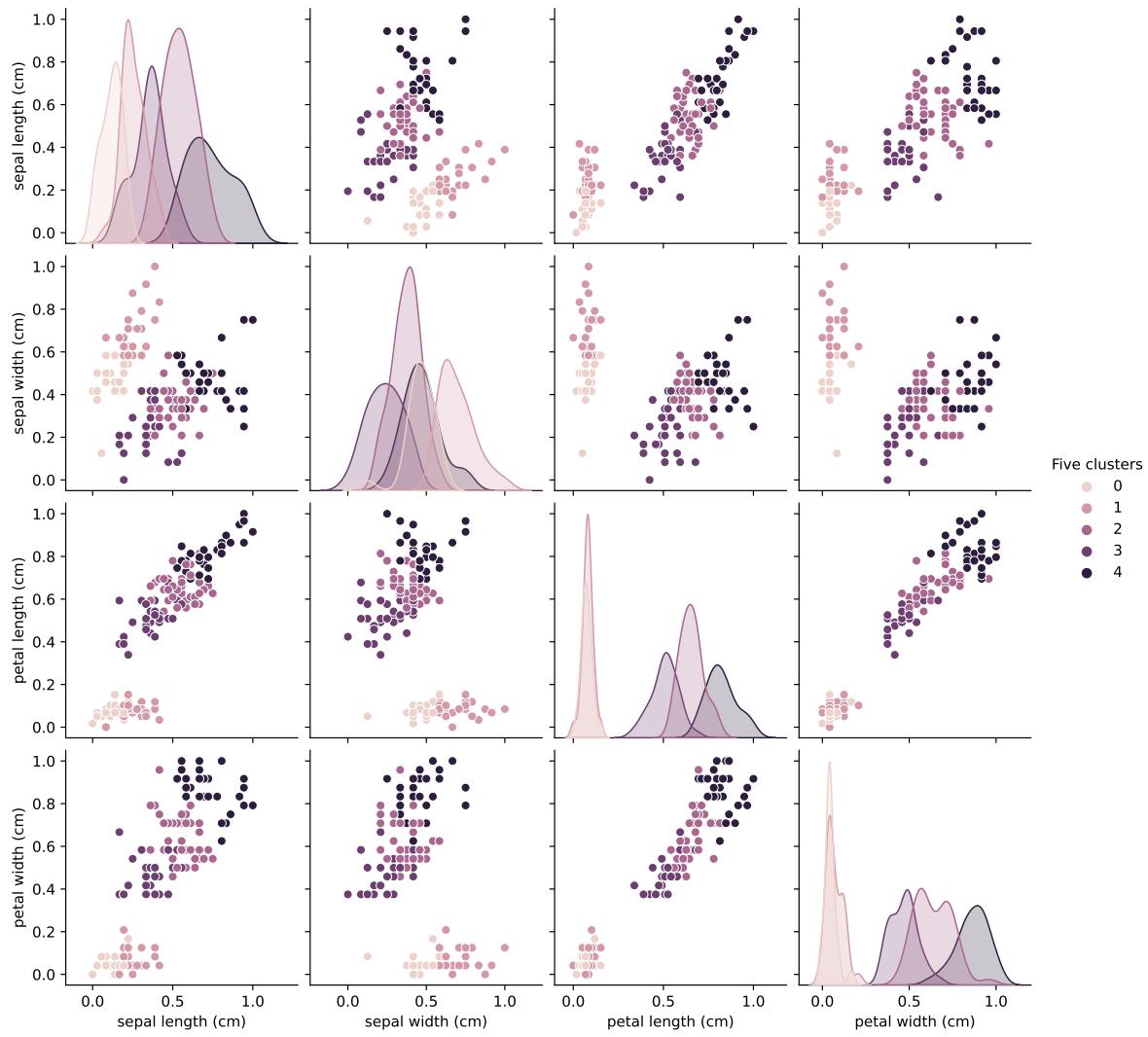


Perhaps we should try 5 clusters instead.

```
k_means_5 = KMeans(n_clusters = 5, init = 'random', n_init = 10)
k_means_5.fit(iris)
iris_df['Five clusters'] = pd.Series(k_means_5.predict(iris_df.iloc[:,0:4].values), index
```

Plot without the columns called 'cluster' and 'Two cluster'

```
sns.pairplot(iris_df.loc[:, (iris_df.columns != 'Three clusters') & (iris_df.columns != 'T
```



`iris_df`

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Three clusters	Two cluster
0	0.222222	0.625000	0.067797	0.041667	0	0
1	0.166667	0.416667	0.067797	0.041667	0	0
2	0.111111	0.500000	0.050847	0.041667	0	0
3	0.083333	0.458333	0.084746	0.041667	0	0
4	0.194444	0.666667	0.067797	0.041667	0	0
...
145	0.666667	0.416667	0.711864	0.916667	2	1

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Three clusters	Two clusters
146	0.555556	0.208333	0.677966	0.750000	1	1
147	0.611111	0.416667	0.711864	0.791667	2	1
148	0.527778	0.583333	0.745763	0.916667	2	1
149	0.444444	0.416667	0.694915	0.708333	1	1

Which did best?

```
k_means.inertia_
```

6.982216473785234

```
k_means_2.inertia_
```

12.127790750538193

```
k_means_5.inertia_
```

4.580323115230179

It looks like our $k = 5$ model captures the data well. Intertia, [looking at the sklearn documentation](#) as the *Sum of squared distances of samples to their closest cluster center..*

If you want to dive further into this then Real Python's [practical guide to K-Means Clustering](#) is quite good.

18.6 Principal Component Analysis (PCA)

PCA reduces the dimension of our data. The method derives point in an n dimensional space from our data which are uncorrelated.

To carry out a PCA on our Iris dataset where there are only two dimensions.

```
from sklearn.decomposition import PCA
```

```
n_components = 2
```

```
pca = PCA(n_components=n_components)
iris_pca = pca.fit(iris_df.iloc[:,0:4])
```

We can look at the components.

```
iris_pca.components_

array([[ 0.42494212, -0.15074824,  0.61626702,  0.64568888],
       [ 0.42320271,  0.90396711, -0.06038308, -0.00983925]])
```

These components are interesting. You may want to look at a [PennState article on interpreting PCA components](#).

Our second column, ‘sepal width (cm)’ is positively correlated with our second principle component whereas the first column ‘sepal length (cm)’ is positively correlated with both.

You may want to consider:

- Do we need more than two components?
- Is it useful to keep sepal length (cm) in the dataset?

We can also examine the explained variance of the each principle component.

```
iris_pca.explained_variance_

array([0.23245325,  0.0324682 ])
```

A nice worked example showing the link between the explained variance and the component is [here](#).

Our first principle component explains a lot more of the variance of data then the second.

18.6.1 Dimension reduction

For our purposes, we are interested in using PCA for reducing the number of dimension in our data whilst preseving the maximal data variance.

We can extract the projected components from the model.

```
iris_pca_vals = pca.fit_transform(iris_df.iloc[:,0:4])
```

The numpy arrays contains the projected values.

```
type(iris_pca_vals)

numpy.ndarray

iris_pca_vals

array([[-6.30702931e-01,  1.07577910e-01],
       [-6.22904943e-01, -1.04259833e-01],
       [-6.69520395e-01, -5.14170597e-02],
       [-6.54152759e-01, -1.02884871e-01],
       [-6.48788056e-01,  1.33487576e-01],
       [-5.35272778e-01,  2.89615724e-01],
       [-6.56537790e-01,  1.07244911e-02],
       [-6.25780499e-01,  5.71335411e-02],
       [-6.75643504e-01, -2.00703283e-01],
       [-6.45644619e-01, -6.72080097e-02],
       [-5.97408238e-01,  2.17151953e-01],
       [-6.38943190e-01,  3.25988375e-02],
       [-6.61612593e-01, -1.15605495e-01],
       [-7.51967943e-01, -1.71313322e-01],
       [-6.00371589e-01,  3.80240692e-01],
       [-5.52157227e-01,  5.15255982e-01],
       [-5.77053593e-01,  2.93709492e-01],
       [-6.03799228e-01,  1.07167941e-01],
       [-5.20483461e-01,  2.87627289e-01],
       [-6.12197555e-01,  2.19140388e-01],
       [-5.57674300e-01,  1.02109180e-01],
       [-5.79012675e-01,  1.81065123e-01],
       [-7.37784662e-01,  9.05588211e-02],
       [-5.06093857e-01,  2.79470846e-02],
       [-6.07607579e-01,  2.95285112e-02],
       [-5.90210587e-01, -9.45510863e-02],
       [-5.61527888e-01,  5.52901611e-02],
       [-6.08453780e-01,  1.18310099e-01],
       [-6.12617807e-01,  8.16682448e-02],
       [-6.38184784e-01, -5.44873860e-02],
       [-6.20099660e-01, -8.03970516e-02],
       [-5.24757301e-01,  1.03336126e-01],
       [-6.73044544e-01,  3.44711846e-01],
       [-6.27455379e-01,  4.18257508e-01],
```

$[-6.18740916e-01, -6.76179787e-02]$,
 $[-6.44553756e-01, -1.51267253e-02]$,
 $[-5.93932344e-01, 1.55623876e-01]$,
 $[-6.87495707e-01, 1.22141914e-01]$,
 $[-6.92369885e-01, -1.62014545e-01]$,
 $[-6.13976551e-01, 6.88891719e-02]$,
 $[-6.26048380e-01, 9.64357527e-02]$,
 $[-6.09693996e-01, -4.14325957e-01]$,
 $[-7.04932239e-01, -8.66839521e-02]$,
 $[-5.14001659e-01, 9.21355196e-02]$,
 $[-5.43513037e-01, 2.14636651e-01]$,
 $[-6.07805187e-01, -1.16425433e-01]$,
 $[-6.28656055e-01, 2.18526915e-01]$,
 $[-6.70879139e-01, -6.41961326e-02]$,
 $[-6.09212186e-01, 2.05396323e-01]$,
 $[-6.29944525e-01, 2.04916869e-02]$,
 $[2.79951766e-01, 1.79245790e-01]$,
 $[2.15141376e-01, 1.10348921e-01]$,
 $[3.22223106e-01, 1.27368010e-01]$,
 $[5.94030131e-02, -3.28502275e-01]$,
 $[2.62515235e-01, -2.95800761e-02]$,
 $[1.03831043e-01, -1.21781742e-01]$,
 $[2.44850362e-01, 1.33801733e-01]$,
 $[-1.71529386e-01, -3.52976762e-01]$,
 $[2.14230599e-01, 2.06607890e-02]$,
 $[1.53249619e-02, -2.12494509e-01]$,
 $[-1.13710323e-01, -4.93929201e-01]$,
 $[1.37348380e-01, -2.06894998e-02]$,
 $[4.39928190e-02, -3.06159511e-01]$,
 $[1.92559767e-01, -3.95507760e-02]$,
 $[-8.26091518e-03, -8.66610981e-02]$,
 $[2.19485489e-01, 1.09383928e-01]$,
 $[1.33272148e-01, -5.90267184e-02]$,
 $[-5.75757060e-04, -1.42367733e-01]$,
 $[2.54345249e-01, -2.89815304e-01]$,
 $[-5.60800300e-03, -2.39572672e-01]$,
 $[2.68168358e-01, 4.72705335e-02]$,
 $[9.88208151e-02, -6.96420088e-02]$,
 $[2.89086481e-01, -1.69157553e-01]$,
 $[1.45033538e-01, -7.63961345e-02]$,
 $[1.59287093e-01, 2.19853643e-04]$,
 $[2.13962718e-01, 5.99630005e-02]$,
 $[2.91913782e-01, 4.04990109e-03]$,

```

[ 3.69148997e-01,  6.43480720e-02] ,
[ 1.86769115e-01, -4.96694916e-02] ,
[-6.87697501e-02, -1.85648007e-01] ,
[-2.15759776e-02, -2.87970157e-01] ,
[-5.89248844e-02, -2.86536746e-01] ,
[ 3.23412419e-02, -1.41140786e-01] ,
[ 2.88906394e-01, -1.31550706e-01] ,
[ 1.09664252e-01, -8.25379800e-02] ,
[ 1.82266934e-01,  1.38247021e-01] ,
[ 2.77724803e-01,  1.05903632e-01] ,
[ 1.95615410e-01, -2.38550997e-01] ,
[ 3.76839264e-02, -5.41130122e-02] ,
[ 4.68406593e-02, -2.53171683e-01] ,
[ 5.54365941e-02, -2.19190186e-01] ,
[ 1.75833387e-01, -8.62037590e-04] ,
[ 4.90676225e-02, -1.79829525e-01] ,
[-1.53444261e-01, -3.78886428e-01] ,
[ 6.69726607e-02, -1.68132343e-01] ,
[ 3.30293747e-02, -4.29708545e-02] ,
[ 6.62142547e-02, -8.10461198e-02] ,
[ 1.35679197e-01, -2.32914079e-02] ,
[-1.58634575e-01, -2.89139847e-01] ,
[ 6.20502279e-02, -1.17687974e-01] ,
[ 6.22771338e-01,  1.16807265e-01] ,
[ 3.46009609e-01, -1.56291874e-01] ,
[ 6.17986434e-01,  1.00519741e-01] ,
[ 4.17789309e-01, -2.68903690e-02] ,
[ 5.63621248e-01,  3.05994289e-02] ,
[ 7.50122599e-01,  1.52133800e-01] ,
[ 1.35857804e-01, -3.30462554e-01] ,
[ 6.08945212e-01,  8.35018443e-02] ,
[ 5.11020215e-01, -1.32575915e-01] ,
[ 7.20608541e-01,  3.34580389e-01] ,
[ 4.24135062e-01,  1.13914054e-01] ,
[ 4.37723702e-01, -8.78049736e-02] ,
[ 5.40793776e-01,  6.93466165e-02] ,
[ 3.63226514e-01, -2.42764625e-01] ,
[ 4.74246948e-01, -1.20676423e-01] ,
[ 5.13932631e-01,  9.88816323e-02] ,
[ 4.24670824e-01,  3.53096310e-02] ,
[ 7.49026039e-01,  4.63778390e-01] ,
[ 8.72194272e-01,  9.33798117e-03] ,
[ 2.82963372e-01, -3.18443776e-01] ,

```

```
[ 6.14733184e-01,  1.53566018e-01],  
[ 3.22133832e-01, -1.40500924e-01],  
[ 7.58030401e-01,  8.79453649e-02],  
[ 3.57235237e-01, -9.50568671e-02],  
[ 5.31036706e-01,  1.68539991e-01],  
[ 5.46962123e-01,  1.87812429e-01],  
[ 3.28704908e-01, -6.81237595e-02],  
[ 3.14783811e-01, -5.57223965e-03],  
[ 5.16585543e-01, -5.40299414e-02],  
[ 4.84826663e-01,  1.15348658e-01],  
[ 6.33043632e-01,  5.92290940e-02],  
[ 6.87490917e-01,  4.91179916e-01],  
[ 5.43489246e-01, -5.44399104e-02],  
[ 2.91133358e-01, -5.82085481e-02],  
[ 3.05410131e-01, -1.61757644e-01],  
[ 7.63507935e-01,  1.68186703e-01],  
[ 5.47805644e-01,  1.58976299e-01],  
[ 4.06585699e-01,  6.12192966e-02],  
[ 2.92534659e-01, -1.63044284e-02],  
[ 5.35871344e-01,  1.19790986e-01],  
[ 6.13864965e-01,  9.30029331e-02],  
[ 5.58343139e-01,  1.22041374e-01],  
[ 3.46009609e-01, -1.56291874e-01],  
[ 6.23819644e-01,  1.39763503e-01],  
[ 6.38651518e-01,  1.66900115e-01],  
[ 5.51461624e-01,  5.98413741e-02],  
[ 4.07146497e-01, -1.71820871e-01],  
[ 4.47142619e-01,  3.75600193e-02],  
[ 4.88207585e-01,  1.49677521e-01],  
[ 3.12066323e-01, -3.11303854e-02]])
```

Each row corresponds to a row in our data.

```
iris_pca_vals.shape
```

```
(150, 2)
```

```
iris_df.shape
```

```
(150, 7)
```

We can add the component to our dataset. I prefer to keep everything in one table and it is not at all required. You can just assign the values whichever variables you prefer.

```
iris_df['c1'] = [item[0] for item in iris_pca_vals]
iris_df['c2'] = [item[1] for item in iris_pca_vals]
```

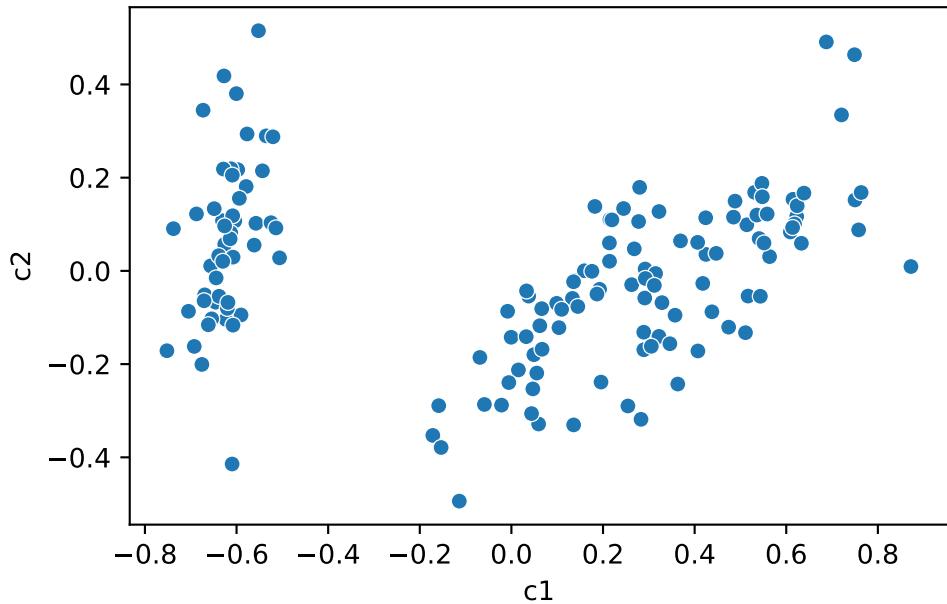
```
iris_df
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Three clusters	Two cluster
0	0.222222	0.625000	0.067797	0.041667	0	0
1	0.166667	0.416667	0.067797	0.041667	0	0
2	0.111111	0.500000	0.050847	0.041667	0	0
3	0.083333	0.458333	0.084746	0.041667	0	0
4	0.194444	0.666667	0.067797	0.041667	0	0
...
145	0.666667	0.416667	0.711864	0.916667	2	1
146	0.555556	0.208333	0.677966	0.750000	1	1
147	0.611111	0.416667	0.711864	0.791667	2	1
148	0.527778	0.583333	0.745763	0.916667	2	1
149	0.444444	0.416667	0.694915	0.708333	1	1

Plotting out our data on our new two component space.

```
sns.scatterplot(data = iris_df, x = 'c1', y = 'c2')
```

```
<Axes: xlabel='c1', ylabel='c2'>
```

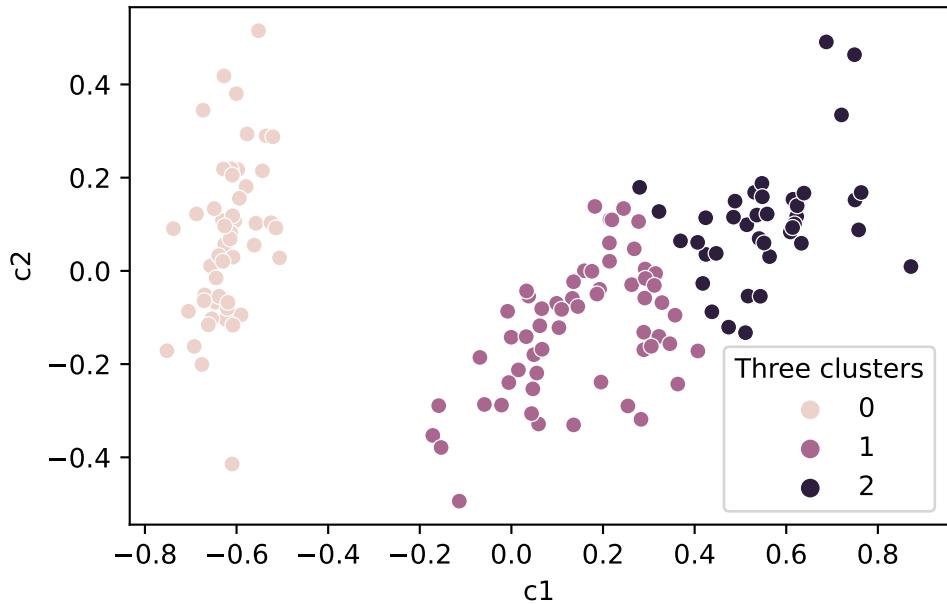


We have reduced our three dimensions to two.

We can also colour by our clusters. What does this show us and is it useful?

```
sns.scatterplot(data = iris_df, x = 'c1', y = 'c2', hue = 'Three clusters')

<Axes: xlabel='c1', ylabel='c2'>
```



iris_df

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Three clusters	Two clusters
0	0.222222	0.625000	0.067797	0.041667	0	0
1	0.166667	0.416667	0.067797	0.041667	0	0
2	0.111111	0.500000	0.050847	0.041667	0	0
3	0.083333	0.458333	0.084746	0.041667	0	0
4	0.194444	0.666667	0.067797	0.041667	0	0
...
145	0.666667	0.416667	0.711864	0.916667	2	1
146	0.555556	0.208333	0.677966	0.750000	1	1
147	0.611111	0.416667	0.711864	0.791667	2	1
148	0.527778	0.583333	0.745763	0.916667	2	1
149	0.444444	0.416667	0.694915	0.708333	1	1

18.7 PCA to Clusters

We have reduced our 4D dataset to 2D whilst keeping the data variance. Reducing the data to fewer dimensions can help with the ‘curse of dimensionality’, reduce the chance of overfitting a machine learning model (see [here](#)) and reduce the computational complexity of a model fit.

Putting our new dimensions into a kMeans model

```
k_means_pca = KMeans(n_clusters = 3, init = 'random', n_init = 10)
iris_pca_kmeans = k_means_pca.fit(iris_df.iloc[:, -2:])

type(iris_df.iloc[:, -2:].values)

numpy.ndarray

iris_df['PCA 3 clusters'] = pd.Series(k_means_pca.predict(iris_df.iloc[:, -2:].values), index=iris_df.index)
```

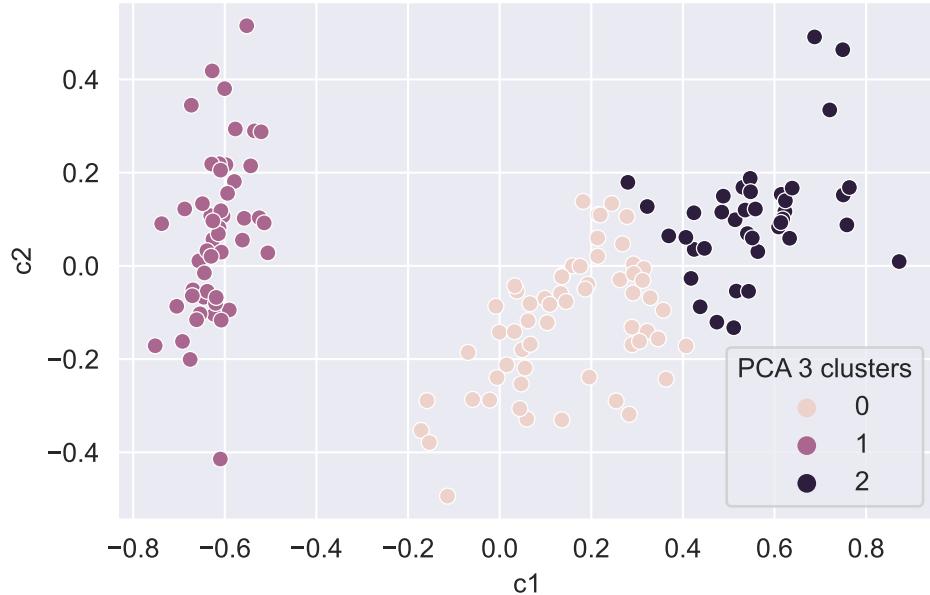
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/sklearn/base.py:464: UserWarning:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Three clusters	Two clusters
0	0.222222	0.625000	0.067797	0.041667	0	0
1	0.166667	0.416667	0.067797	0.041667	0	0
2	0.111111	0.500000	0.050847	0.041667	0	0
3	0.083333	0.458333	0.084746	0.041667	0	0
4	0.194444	0.666667	0.067797	0.041667	0	0
...
145	0.666667	0.416667	0.711864	0.916667	2	1
146	0.555556	0.208333	0.677966	0.750000	1	1
147	0.611111	0.416667	0.711864	0.791667	2	1
148	0.527778	0.583333	0.745763	0.916667	2	1
149	0.444444	0.416667	0.694915	0.708333	1	1

As we only have two dimensions we can easily plot this on a single scatterplot.

```
# a different seaborn theme
# see https://python-graph-gallery.com/104-seaborn-themes/
sns.set_style("darkgrid")
sns.scatterplot(data = iris_df, x = 'c1', y = 'c2', hue = 'PCA 3 clusters')
```

<Axes: xlabel='c1', ylabel='c2'>



I suspect having two clusters would work better. We should try a few different models.

Copying the code from [here](#) we can fit multiple numbers of clusters.

```

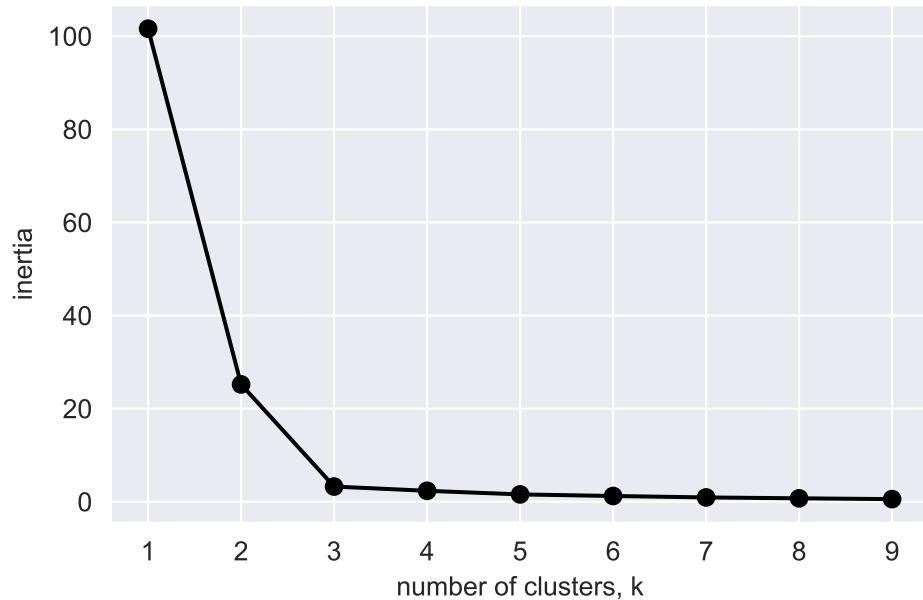
ks = range(1, 10)
inertias = [] # Create an empty list (will be populated later)
for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k, n_init = 10)

    # Fit model to samples
    model.fit(iris_df.iloc[:, -2:])

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

plt.plot(ks, inertias, '-o', color='black')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()

```



Three seems ok. We clearly want no more than three.

These types of plots show an point about model complexity. More free parameters in the model (here the number of clusters) will improve how well the model captures the data, often with reducing returns. However, a model which overfits the data will not be able to fit new data well - referred to overfitting. Randomish internet blogs introduce the topic pretty well, see [here](#), and also wikipedia, see [here](#).

18.8 Missing values

Finally, how we deal with missing values can impact the results of PCA and kMeans clustering.

Lets us load in the iris dataset again and randomly remove 10% of the data (see code from [here](#)).

```
import numpy as np

x = load_iris()

iris_df = pd.DataFrame(x.data, columns = x.feature_names)
```

```
mask = np.random.choice([True, False], size = iris_df.shape, p = [0.2, 0.8])
mask[mask.all(1), -1] = 0
```

```
df = iris_df.mask(mask)
```

```
df.isna().sum()
```

```
sepal length (cm)    37
sepal width (cm)    33
petal length (cm)   28
petal width (cm)    26
dtype: int64
```

```
df
```

	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	NaN
2	4.7	3.2	1.3	NaN
3	4.6	3.1	1.5	0.2
4	NaN	3.6	1.4	0.2
...
145	NaN	NaN	NaN	2.3
146	NaN	2.5	NaN	1.9
147	6.5	3.0	5.2	NaN
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	NaN

About 20% of the data is randomly an NaN.

18.8.1 Zeroing

We can 0 them and fit our models.

```
df_1 = df.copy()
df_1 = df_1.fillna(0)
```

```
df_1
```

	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	0.0
2	4.7	3.2	1.3	0.0
3	4.6	3.1	1.5	0.2
4	0.0	3.6	1.4	0.2
...
145	0.0	0.0	0.0	2.3
146	0.0	2.5	0.0	1.9
147	6.5	3.0	5.2	0.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	0.0

```

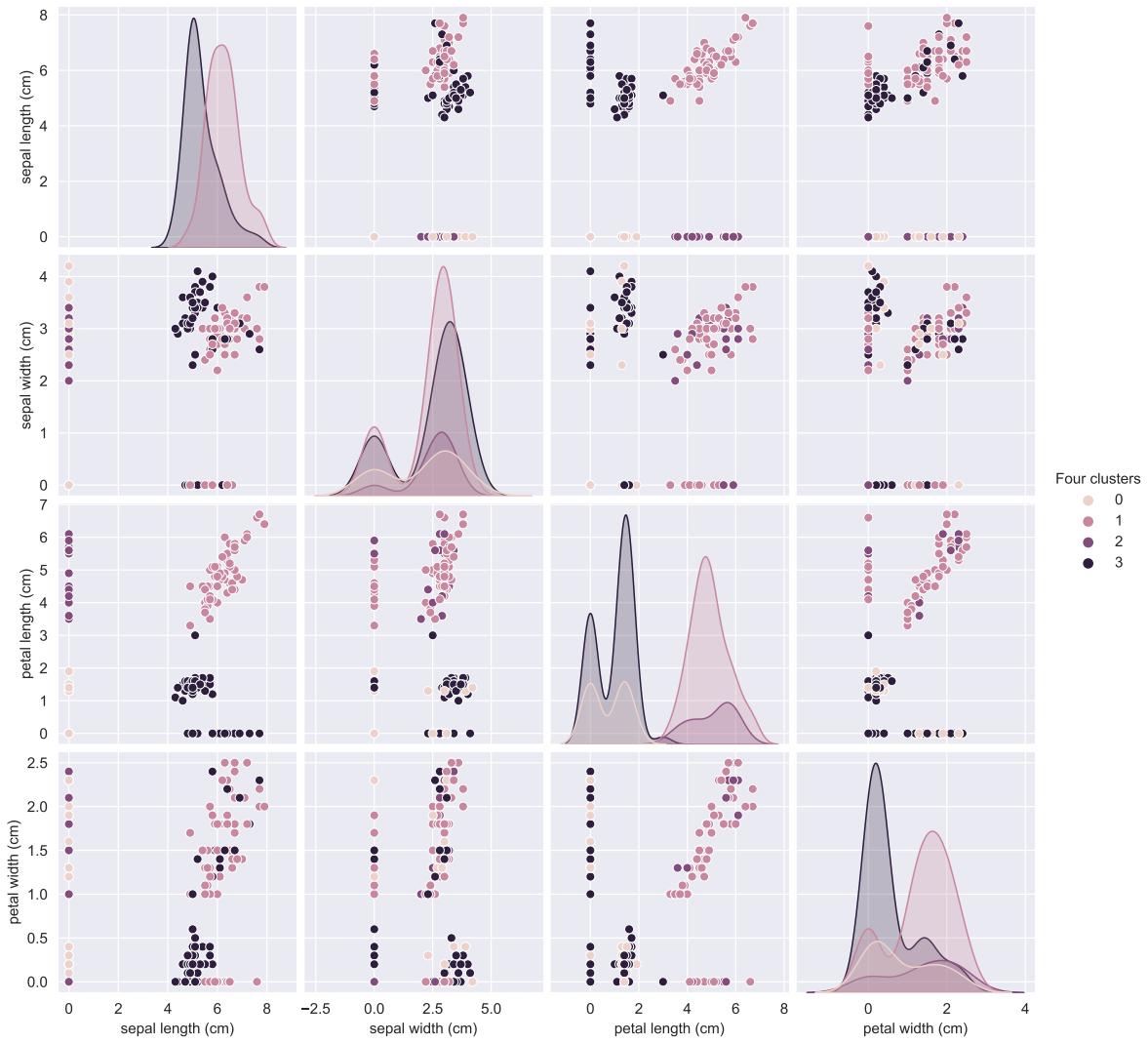
k_means_zero = KMeans(n_clusters = 4, init = 'random', n_init = 10)
k_means_zero.fit(df_1)
df_1['Four clusters'] = pd.Series(k_means_zero.predict(df_1.iloc[:,0:4].values), index = df_1.index)
sns.pairplot(df_1, hue = 'Four clusters')

```

```

/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/sklearn/base.py:464: UserWarning:
  warnings.warn(

```



What impact has zeroing the values had on our results?

Now, onto PCA.

```
# PCA analysis
n_components = 2

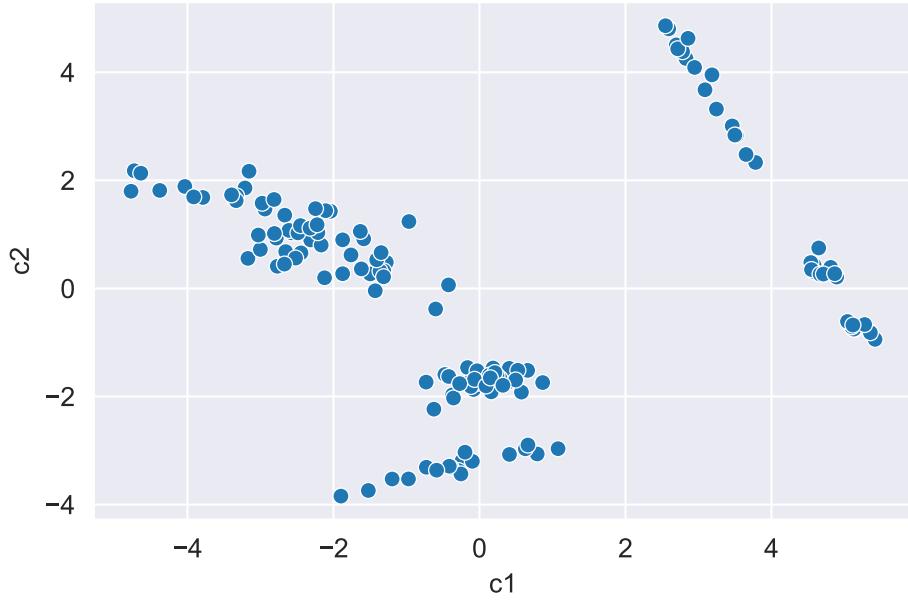
pca = PCA(n_components=n_components)
df_1_pca = pca.fit(df_1.iloc[:,0:4])

# Extract projected values
```

```
df_1_pca_vals = df_1_pca.transform(df_1.iloc[:,0:4])
df_1['c1'] = [item[0] for item in df_1_pca_vals]
df_1['c2'] = [item[1] for item in df_1_pca_vals]

sns.scatterplot(data = df_1, x = 'c1', y = 'c2')
```

```
<Axes: xlabel='c1', ylabel='c2'>
```



```
df_1_pca.explained_variance_
```

```
array([7.43333957, 4.22999611])
```

```
df_1_pca.components_
```

```
array([[-0.90610821, -0.07575768, -0.41005781, -0.07128304],
       [-0.41852132,  0.01814027,  0.89754426,  0.13756864]])
```

18.8.2 Replacing with the average

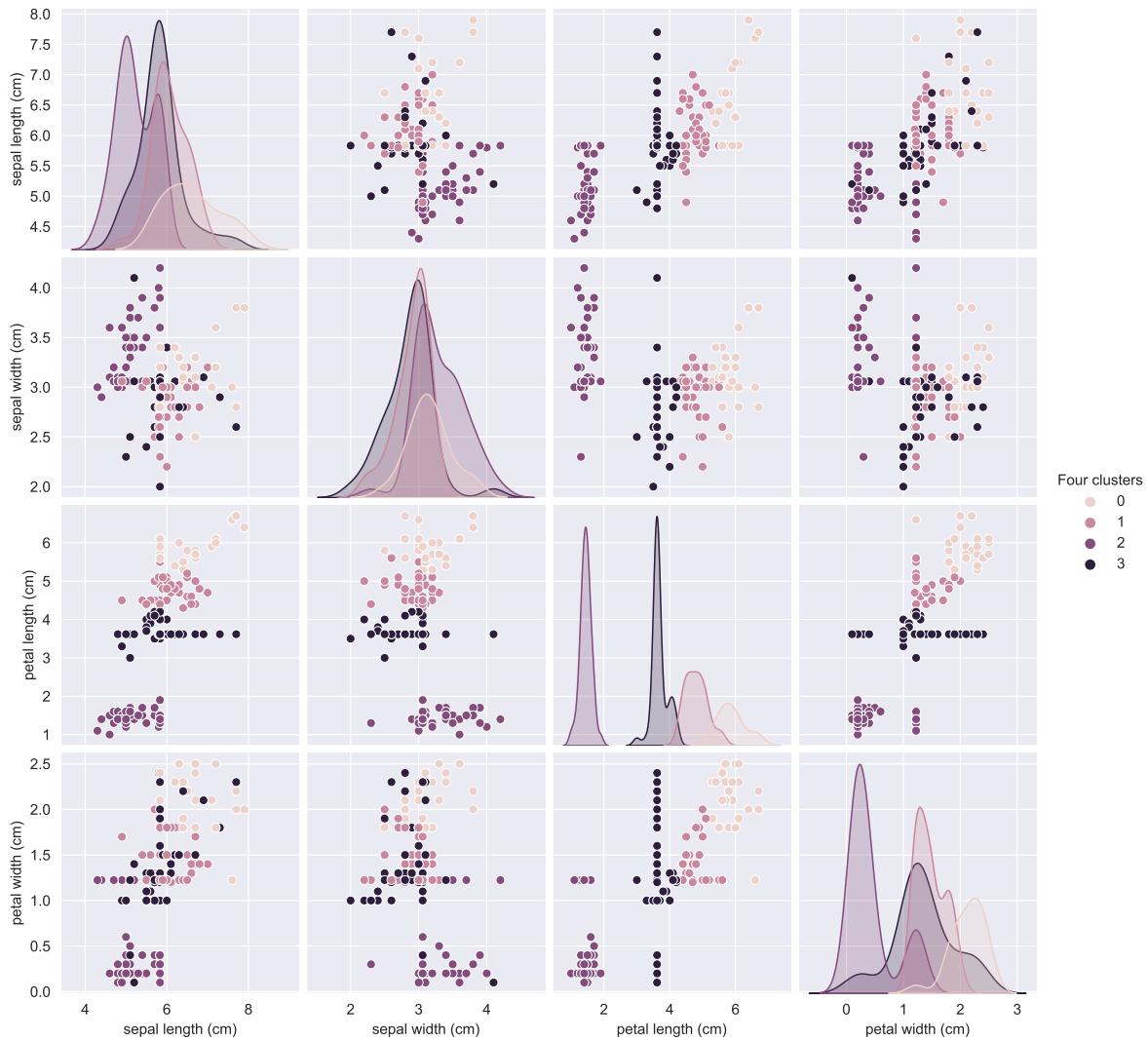
```
df_2 = df.copy()
for i in range(4):
    df_2.iloc[:,i] = df_2.iloc[:,i].fillna(df_2.iloc[:,i].mean())
```

```
df_2
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.100000	3.500000	1.400000	0.200000
1	4.900000	3.000000	1.400000	1.224194
2	4.700000	3.200000	1.300000	1.224194
3	4.600000	3.100000	1.500000	0.200000
4	5.832743	3.600000	1.400000	0.200000
...
145	5.832743	3.058974	3.616393	2.300000
146	5.832743	2.500000	3.616393	1.900000
147	6.500000	3.000000	5.200000	1.224194
148	6.200000	3.400000	5.400000	2.300000
149	5.900000	3.000000	5.100000	1.224194

```
k_means_zero = KMeans(n_clusters = 4, init = 'random')
k_means_zero.fit(df_2)
df_2['Four clusters'] = pd.Series(k_means_zero.predict(df_2.iloc[:,0:4].values), index = df_2.index)
sns.pairplot(df_2, hue = 'Four clusters')
```

```
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/sklearn/cluster/_kmeans.py:464: UserWarning: The number of clusters (4) is greater than the number of samples (150). In this case, KMeans will fall back to full centroid
super().__check_params_vs_input(X, default_n_init=10)
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/sklearn/base.py:464: UserWarning: The number of clusters (4) is greater than the number of samples (150). In this case, KMeans will fall back to full centroid
```



```

# PCA analysis
n_components = 2

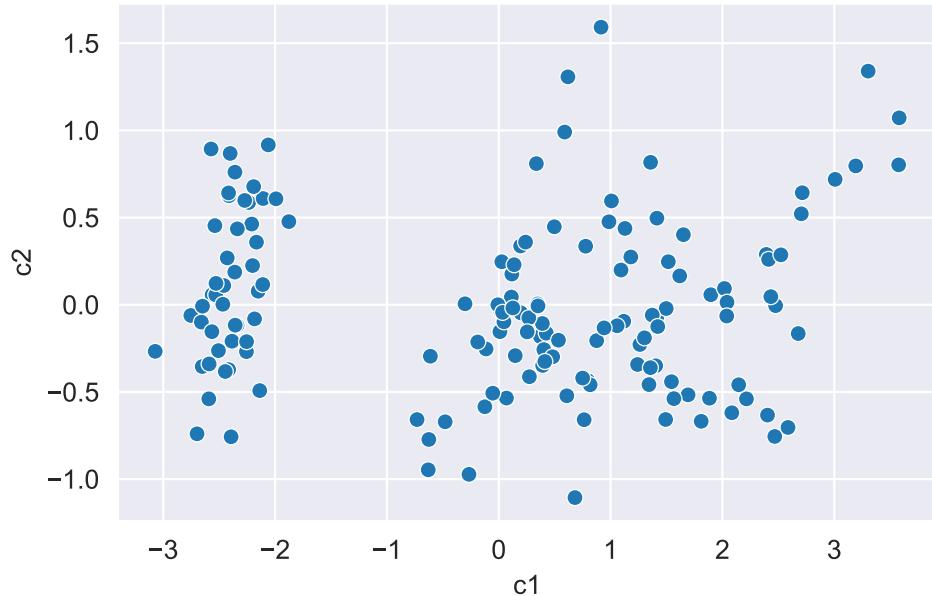
pca = PCA(n_components=n_components)
df_2_pca = pca.fit(df_2.iloc[:,0:4])

# Extract projected values
df_2_pca_vals = df_2_pca.transform(df_2.iloc[:,0:4])
df_2['c1'] = [item[0] for item in df_2_pca_vals]
df_2['c2'] = [item[1] for item in df_2_pca_vals]

```

```
sns.scatterplot(data = df_2, x = 'c1', y = 'c2')
```

```
<Axes: xlabel='c1', ylabel='c2'>
```



```
df_2_pca.explained_variance_
```

```
array([3.27059853, 0.24512038])
```

```
df_2_pca.components_
```

```
array([[ 0.28700466, -0.06057097,  0.89861739,  0.32626106],
       [ 0.92272106,  0.27081109, -0.27424507, -0.00607048]])
```

19 Useful resources

The scikit learn UserGuide is very good. Both approaches here are often referred to as unsupervised learning methods and you can find the scikit learn section on these [here](#).

If you have issues with the documentation then also look at the scikit-learn [examples](#).

Also, in no particular order:

- The [In-Depth sections of the Python Data Science Handbook](#). More for machine learning but interesting all the same.
- [Python for Data Analysis](#) (ebook is available via [Warwick library](#))

In case you are bored:

- [Stack abuse](#) - Some fun blog entries to look at
- [Towards data science](#) - a blog that contains a mix of intro, intermediate and advanced topics. Nice to skim through to try and understand something new.

Please do try out some of the techniques detailed in the lecture material. The simple examples found in the scikit learn documentation are rather good. Generally, I find it much easier to try to understand a method using a simple dataset.

20 IM939 Lab 4 - Part 2 - Crime

Below we work with a subset of the US Census & Crime data. Details about the various variables can be found [here](#).

How can we tackle a dataset with 100 different dimensions? We try using PCA and clustering. There is a lot you can do with this dataset. We encourage you to dig in. Try and find some interesting patterns in the data. Even upload your findings to the Teams channel and discuss.

```
import pandas as pd
df = pd.read_csv('data/censusCrimeClean.csv')
```

```
df
```

	communityname	fold	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	Lakewoodcity	1	0.19	0.33	0.02	0.90	0.12
1	Tukwilacity	1	0.00	0.16	0.12	0.74	0.45
2	Aberdeentown	1	0.00	0.42	0.49	0.56	0.17
3	Willingborotownship	1	0.04	0.77	1.00	0.08	0.12
4	Bethlehemptownship	1	0.01	0.55	0.02	0.95	0.09
...
1989	TempleTerracecity	10	0.01	0.40	0.10	0.87	0.12
1990	Seasidecity	10	0.05	0.96	0.46	0.28	0.83
1991	Waterburytown	10	0.16	0.37	0.25	0.69	0.04
1992	Walthamcity	10	0.08	0.51	0.06	0.87	0.22
1993	Ontariocity	10	0.20	0.78	0.14	0.46	0.24

```
from sklearn.decomposition import PCA

n_components = 2

pca = PCA(n_components=n_components)
df_pca = pca.fit(df.iloc[:, 1:])
```

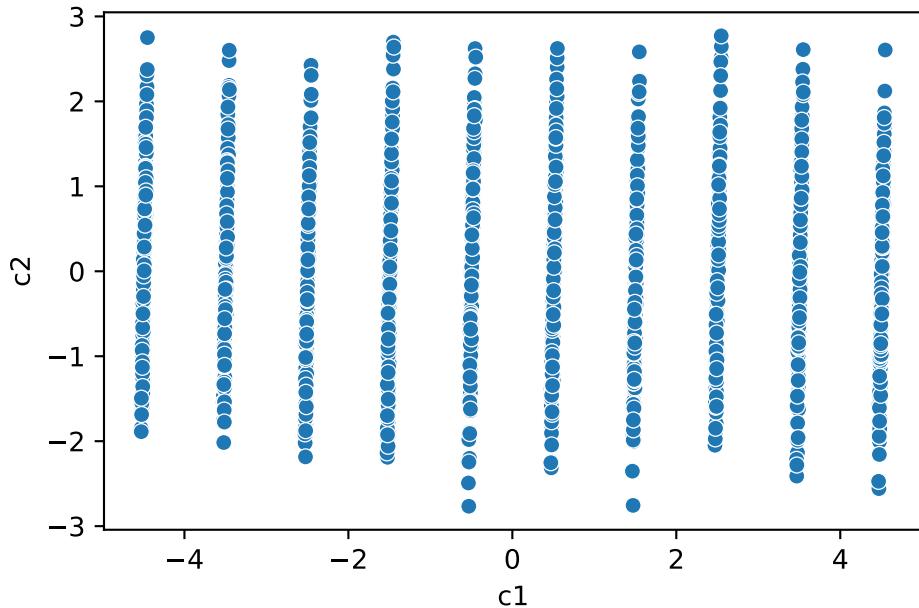
```
df_pca.explained_variance_
```

```
array([8.26130789, 1.08655844])
```

```
df_pca_vals = pca.fit_transform(df.iloc[:,1:])
df['c1'] = [item[0] for item in df_pca_vals]
df['c2'] = [item[1] for item in df_pca_vals]
```

```
import seaborn as sns
sns.scatterplot(data = df, x = 'c1', y = 'c2')
```

```
<Axes: xlabel='c1', ylabel='c2'>
```



Hmm, that looks problematic. What is going on?

We should look at the component loadings. Though there are going to be over 200 of them. We can put them into a Pandas dataframe and then sort them.

```

data = {'columns' : df.columns[1:102],
        'component 1' : df_pca.components_[0],
        'component 2' : df_pca.components_[1]}

loadings = pd.DataFrame(data)
loadings.sort_values(by=['component 1'], ascending=False)

```

	columns	component 1	component 2
0	fold	0.999788	-0.016430
12	pctUrban	0.004944	0.146079
85	RentHighQ	0.004078	0.188187
13	medIncome	0.003932	0.185649
46	PctYoungKids2Par	0.003818	0.178675
...
17	pctWSocSec	-0.003353	-0.059245
29	PctPopUnderPov	-0.003594	-0.190548
18	pctWPubAsst	-0.003697	-0.173593
10	agePct65up	-0.003767	-0.033010
37	PctOccupManu	-0.004403	-0.132659

```
loadings.sort_values(by=['component 2'], ascending=False)
```

	columns	component 1	component 2
85	RentHighQ	0.004078	0.188187
13	medIncome	0.003932	0.185649
46	PctYoungKids2Par	0.003818	0.178675
20	medFamInc	0.003025	0.177225
45	PctKids2Par	0.002385	0.169581
...
51	PctIlleg	-0.001022	-0.153504
31	PctNotHSGrad	-0.002967	-0.158646
18	pctWPubAsst	-0.003697	-0.173593
29	PctPopUnderPov	-0.003594	-0.190548
78	PctHousNoPhone	-0.001759	-0.198116

The fold variable is messing with our PCA. What does that variable look like.

```
df.fold.value_counts()
```

```
fold
1    200
2    200
3    200
4    200
5    199
6    199
7    199
8    199
9    199
10   199
Name: count, dtype: int64
```

```
df['fold'].unique()
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

Aha! It looks to be some sort of categorical variable. Looking into the dataset more, it is actually a variable used for cross tabling.

We should not include this variable in our analysis.

```
df.iloc[:, 2:-2]
```

	population	householdsize	racepctblack	racePctWhite	racePctAsian	racePctHisp	agePct12t21
0	0.19	0.33	0.02	0.90	0.12	0.17	0.34
1	0.00	0.16	0.12	0.74	0.45	0.07	0.26
2	0.00	0.42	0.49	0.56	0.17	0.04	0.39
3	0.04	0.77	1.00	0.08	0.12	0.10	0.51
4	0.01	0.55	0.02	0.95	0.09	0.05	0.38
...
1989	0.01	0.40	0.10	0.87	0.12	0.16	0.43
1990	0.05	0.96	0.46	0.28	0.83	0.32	0.69
1991	0.16	0.37	0.25	0.69	0.04	0.25	0.35
1992	0.08	0.51	0.06	0.87	0.22	0.10	0.58
1993	0.20	0.78	0.14	0.46	0.24	0.77	0.50

```

from sklearn.decomposition import PCA

n_components = 2

pca_no_fold = PCA(n_components=n_components)
df_pca_no_fold = pca_no_fold.fit(df.iloc[:, 2:-2])

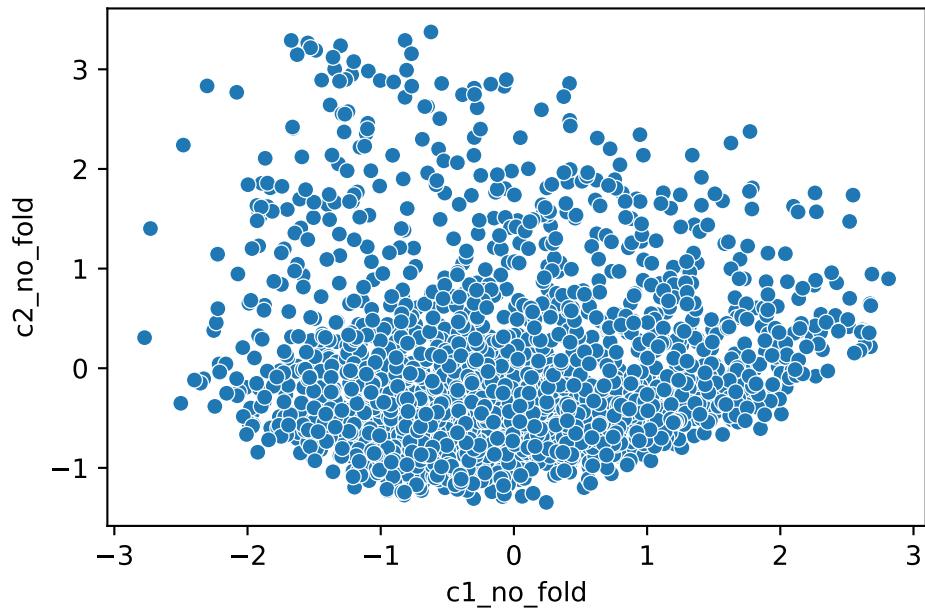
df_pca_vals = pca_no_fold.fit_transform(df.iloc[:, 2:-2])

df['c1_no_fold'] = [item[0] for item in df_pca_vals]
df['c2_no_fold'] = [item[1] for item in df_pca_vals]

sns.scatterplot(data = df, x = 'c1_no_fold', y = 'c2_no_fold')

<Axes: xlabel='c1_no_fold', ylabel='c2_no_fold'>

```



How do our component loadings look?

```

data = {'columns' : df.iloc[:, 2:-4].columns,
        'component 1' : df_pca_no_fold.components_[0],
        'component 2' : df_pca_no_fold.components_[1]}

```

```
loadings = pd.DataFrame(data)
loadings_sorted = loadings.sort_values(by=['component 1'], ascending=False)
loadings_sorted.iloc[1:10, :]
```

	columns	component 1	component 2
12	medIncome	0.185822	0.040774
45	PctYoungKids2Par	0.178807	-0.017237
19	medFamInc	0.177283	0.031895
44	PctKids2Par	0.169533	-0.042621
43	PctFam2Par	0.163277	-0.031094
82	RentLowQ	0.163120	0.119274
85	MedRent	0.163120	0.106135
80	OwnOccMedVal	0.159850	0.135830
83	RentMedian	0.159605	0.113360

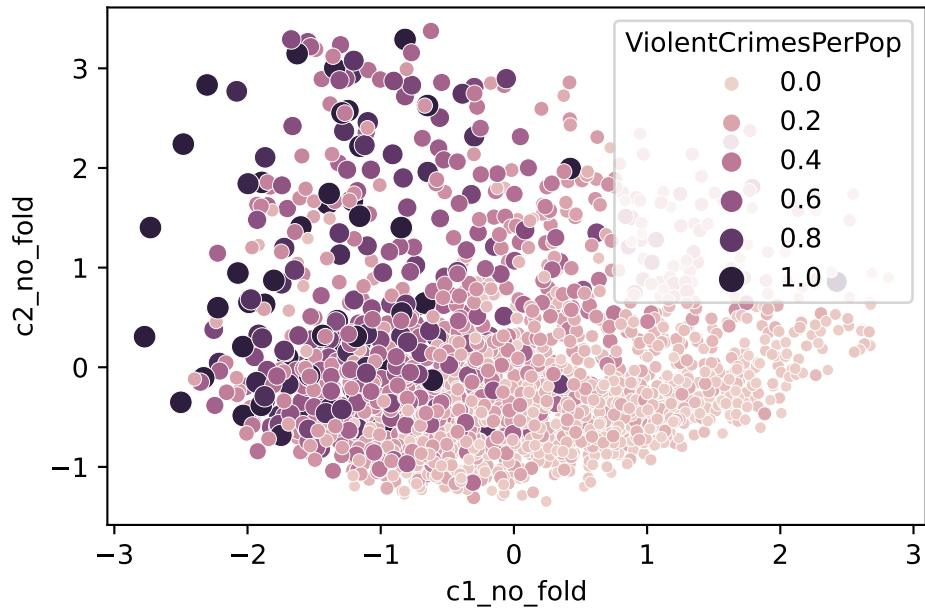
```
loadings_sorted = loadings.sort_values(by=['component 2'], ascending=False)
loadings_sorted.iloc[1:10, :]
```

	columns	component 1	component 2
59	PctRecImmig10	-0.000556	0.253476
57	PctRecImmig5	-0.000901	0.252463
56	PctRecentImmig	0.001797	0.248239
91	PctForeignBorn	0.024743	0.241783
61	PctNotSpeakEnglWell	-0.048595	0.207575
5	racePctHisp	-0.063868	0.187826
68	PctPersDenseHous	-0.087193	0.184704
11	pctUrban	0.146557	0.183317
4	racePctAsian	0.061547	0.160710

Interesting that our first component variables are income related whereas our second component variables are immigration related. If we look at the projections of the model, coloured by Crime, what do we see?

```
sns.scatterplot(data = df,
                 x = 'c1_no_fold',
                 y = 'c2_no_fold',
                 hue = 'ViolentCrimesPerPop',
                 size = 'ViolentCrimesPerPop')
```

```
<Axes: xlabel='c1_no_fold', ylabel='c2_no_fold'>
```



What about clustering using these ‘income’ and ‘immigration’ components?

```
from sklearn.cluster import KMeans

ks = range(1, 10)
inertias = []
for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k, n_init = 10)

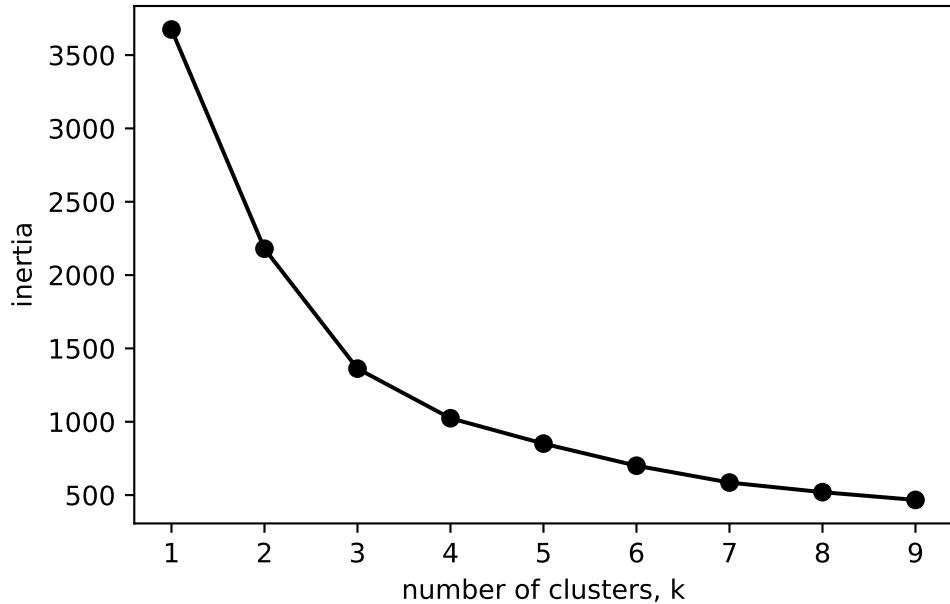
    # Fit model to samples
    model.fit(df[['c1_no_fold', 'c2_no_fold']])

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

import matplotlib.pyplot as plt

plt.plot(ks, inertias, '-o', color='black')
plt.xlabel('number of clusters, k')
```

```
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```

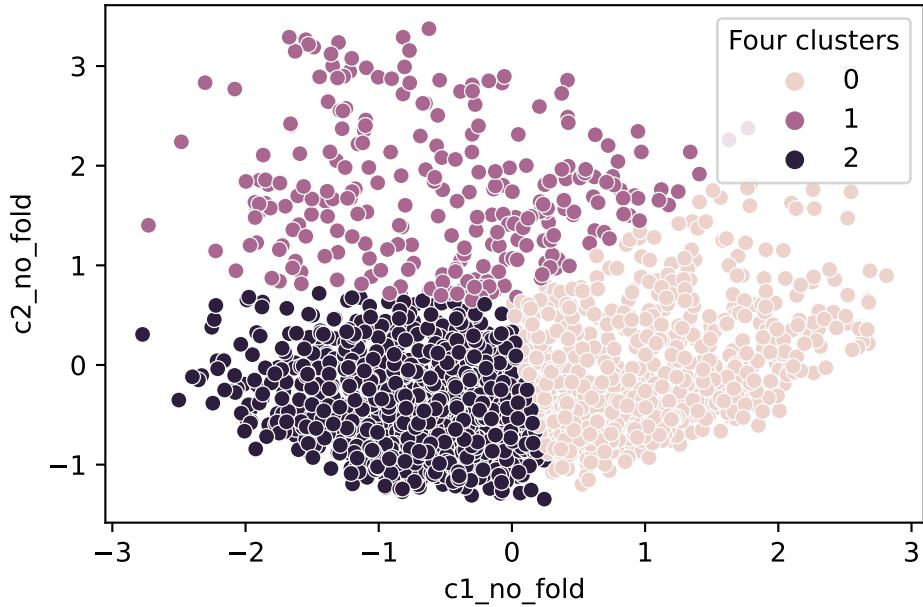


Four clusters looks good.

```
k_means_4 = KMeans(n_clusters = 3, init = 'random', n_init = 10)
k_means_4.fit(df[['c1_no_fold', 'c2_no_fold']])
df['Four clusters'] = pd.Series(k_means_4.predict(df[['c1_no_fold', 'c2_no_fold']]).values)

/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/sklearn/base.py:464: UserWarning
  warnings.warn(
    sns.scatterplot(data = df, x = 'c1_no_fold', y = 'c2_no_fold', hue = 'Four clusters')

<Axes: xlabel='c1_no_fold', ylabel='c2_no_fold'>
```



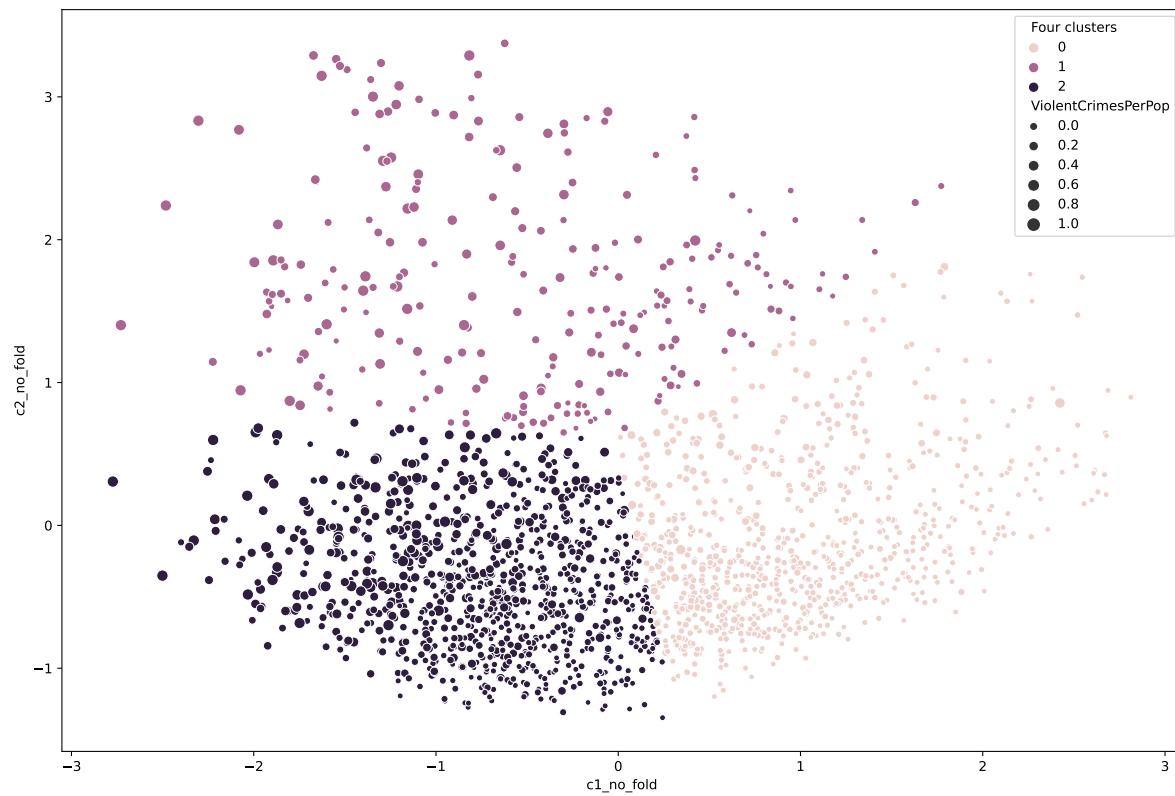
Hmm, might we interpret this plot? The plot is unclear. Let us assume $c1$ is an income component and $c2$ is an immigration component. Cluster 0 are places high on income and low on immigration. Cluster 2 are low on income and low on immigration. The interesting group are those high on immigration and relatively low on income.

We can include crime on the plot.

```
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (15,10)
sns.scatterplot(data = df, x = 'c1_no_fold', y = 'c2_no_fold', hue = 'Four clusters', size=
```

<Axes: xlabel='c1_no_fold', ylabel='c2_no_fold'>



Alas, we stop here in this exercise. You could work on this further. There are outliers you might want to remove. Or, at this stage, you might want to look at more components, 3 or 4 and look for some other factors.

21 IM939 Lab 4 - Part 3 - Exercises

As with previous exercises, fill in the question marks with the correct code.

Last week you were introduced to the [wine dataset](#). We have 10 input variables and 1 output variables.

Input variables (based on physicochemical tests):

- 1 - fixed acidity
- 2 - volatile acidity
- 3 - citric acid
- 4 - residual sugar
- 5 - chlorides
- 6 - free sulfur dioxide
- 7 - total sulfur dioxide
- 8 - density
- 9 - pH
- 10 - sulphates
- 11 - alcohol

Output variable (based on sensory data):

- 12 - quality (score between 0 and 10)

I suggest we look at two broad questions with this dataset:

1. Will dimension reduction reveal variable groupings? Think back to how we interpreted the loadings in the crime dataset.
2. What does clustering the wines tell us?

21.1 Load data and import libraries

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import seaborn as sn?
from sklearn.cluster import KMeans
from sklearn.decomposition import PC?
from sklearn.decomposition import S????ePCA
from sklearn.manifold import TSNE

df = pd.read_excel('data/winequality-red_v2.xlsx')
```

```
SyntaxError: invalid syntax (1138690625.py, line 6)
```

```
df.h??d()
```

```
SyntaxError: invalid syntax (197674394.py, line 1)
```

```
# May take a while depending on your computer
# feel free not to run this
sns.pair????(df)
```

```
SyntaxError: invalid syntax (4010129658.py, line 3)
```

22 Dimension reduction

```
from sklearn.decomposition import PCA  
  
n_components = 2  
  
pca = PCA(n_components=n_components)  
df_pca = pca.fit(df.iloc[:, 0:11])
```

SyntaxError: invalid syntax (2518476773.py, line 5)

```
df_pca_vals = df_pca.transform(df.iloc[:, 0:11])  
df['c1'] = [item[0] for item in df_pca_vals]  
df['c2'] = [item[1] for item in df_pca_vals]
```

SyntaxError: invalid syntax (2292344659.py, line 1)

```
sns.scatterplot(data = df, x = ?, y = ?, hue = 'quality')
```

SyntaxError: invalid syntax (321227352.py, line 1)

```
print(df.columns)  
df_pca.components_
```

NameError: name 'df' is not defined

What about other dimension reduction methods?

22.1 SparsePCA

```
s_pca = SparsePCA(n_components=n_components)
df_s_pca = s_pca.fit(df.????[:, 0:11])
```

```
SyntaxError: invalid syntax (2237240520.py, line 2)
```

```
df_s_pca_vals = s_pca.fit_????????(df.iloc[:, 0:11])
df['c1 spca'] = [item[0] for item in df_s_pca_vals]
df['c2 spca'] = [item[1] for item in df_s_pca_vals]
```

```
SyntaxError: invalid syntax (496583237.py, line 1)
```

```
sns.scatterplot(data = df, x = 'c1 spca', y = 'c2 spca', hue = 'quality')
```

```
NameError: name 'sns' is not defined
```

22.2 tSNE

```
tsne_model = TSNE(n_components=n_components)
df_tsne = tsne_model.fit(df.iloc[:, 0:11])
```

```
NameError: name 'TSNE' is not defined
```

```
df_tsne_vals = tsne_model.fit_transform(df.iloc[:, 0:11])
df['c1 tsne'] = [item[0] for item in ??_tsne_vals]
df['c2 tsne'] = [item[1] for item in df_tsne_vals]
```

```
SyntaxError: invalid syntax (3280343393.py, line 2)
```

```
# This plot does not look right
# I am not sure why.
sns.scatterplot(data = ??, x = 'c1 tsne', y = 'c1 tsne', hue = 'quality')
```

```
SyntaxError: invalid syntax (847552475.py, line 3)
```

That looks concerning - there is a straight line. It looks like something in the data has caused the model to have issues.

Does normalising the data sort out the issue?

```
from sklearn.preprocessing import MinMaxScaler
col_names = df.columns
scaled_df = pd.DataFrame(MinMaxScaler().fit_transform(df))
scaled_df.columns = col_names

NameError: name 'df' is not defined

tsne_model = TSNE(n_components=n_components)

scaled_df_tsne = tsne_model.fit(scaled_df.iloc[:, 0:11])
scaled_df_tsne_vals = tsne_model.fit_transform(df.iloc[:, 0:11])

scaled_df['c1 tsne'] = [item[0] for item in scaled_df_tsne_vals]
scaled_df['c2 tsne'] = [item[1] for item in scaled_df_tsne_vals]

sns.scatterplot(data = scaled_df, x = 'c1 tsne', y = 'c1 tsne', hue = 'quality')
```

```
NameError: name 'TSNE' is not defined
```

Normalising the data makes no difference. It could be the model is getting stuck somehow. You could check the various attributes of the tsne fit object (tsne_model.fit), try using only a few columns and search google a lot - this could be a problem other have encountered.

For now, we will use PCA components.

```
data = {'columns' : df.iloc[:, 0:11].columns,
        'component 1' : df_pca.components_[0],
        'component 2' : df_pca.components_[1]}

loadings = pd.?????????(data)
loadings_sorted = loadings.sort_values(by=['component 1'], ascending=False)
loadings_sorted.iloc[1:10,:]
```

```
SyntaxError: invalid syntax (4262587979.py, line 6)
```

```
loadings_sorted = loadings.sort_values(by=['component 2'], ascending=False)
loadings_sorted.iloc[1:10,:]
```

```
NameError: name 'loadings' is not defined
```

22.3 Clustering

```
from sklearn.cluster import KMeans

ks = range(1, 10)
inertias = []
for k in ks:
    # Create a KMeans instance with k clusters: model
    ????? = KMeans(n_clusters=k)

    # Fit model to samples
    model.fit(df[['c1', 'c2']])

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

import matplotlib.pyplot as plt

plt.plot(ks, inertias, '-o', color='black')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```

```
Object `???? = KMeans(n_clusters=k)` not found.
```

```
NameError: name 'model' is not defined
```

```
k_means_3 = KMeans(n_clusters = 3, init = 'random')
k_means_3.fit(df[['c1', 'c2']])
df['Three clusters'] = pd.Series(k_means_3??????(df[['c1', 'c2']].values), index = df.in
```

```
SyntaxError: invalid syntax (2729138574.py, line 3)
```

```
sns.scatterplot(data = df, x = 'c1', y = 'c2', hue = 'Three clusters')
```

```
NameError: name 'sns' is not defined
```

Consider:

- Is that useful?
- What might it mean?

Outside of this session you could try normalising the data (centering around the mean), clustering the raw data (and not the projections from PCA), trying to get tSNE working or using different numbers of components.

Part VI

Multi-model thinking and rigour

23 IM939 Lab 5 - Part 1

23.1 Workflow

Our labs have focused on data analysis. The goal here is to try and understand informative patterns in our data. These patterns allow us to answer questions.

To do this we:

1. Read data into Python.
2. Look at our data.
3. Wrangling our data. Often exploring raw data and dealing with missing values (imputation techniques), transforming, normalising, standardising, outliers or reshaping.
4. Carry out a series of analysis to better understand our data via clustering, regressions analysis, dimension reduction, and many other techniques.
5. Reflect on what the patterns in our data can tell us.

These are not mutually exclusive processes and are not exhaustive. One may review our data after cleaning, load in more data, carry out additional analysis and/or fit multiple models, tweak data summaries or adopt new techniques. Reflecting on the patterns are in our data can give way to additional analysis and processing.

23.2 So far

A quick reminder, our toolkit comprises of:

- [Pandas](#) - table like data structures. Packed with methods for summarising and manipulating data. [Documentation](#). [Cheat sheet](#).
- [Seaborn](#) - helping us create statistical data visualisations. [Tutorials](#).
- [Scikit-learn](#) - An accessible collection of functions and objects for analysing data. These analysis include dimension reduction, clustering, regressions and evaluating our models. [Examples](#).

These tools comprise some of the core Python data science stack and allow us to tackle many of the elements from each week.

Week 2 Tidy data, data types, wrangling data, imputation ([missing data](#)), transformations.

Week 3 [Descriptive statistics](#), distributions, models (e.g., [regression](#)).

Week 4 Feature selection, dimension reduction (e.g., [Principle Component Analysis](#), [Multidimensional scaling](#), [Linear Discriminant Analysis](#), [t-SNE](#), Correspondance Analysis), [clustering](#) (e.g., [Hierarchical Clustering](#), Partitioning-based clustering such as [K-means](#)).

We have also encountered two dataset sources.

- [sklearn example datasets](#)
- [UCI Machine Learning Repository](#)

You can learn a lot by picking a dataset, choosing a possible research question and carrying a series of analysis. I encourage you to do so outside of the session. It certainly forces one to read the documentation and explore the wonderful possibilities.

23.3 This week

Trying to understand patterns in data often requires us to fit multiple models. We need to consider how well a given model (a kmeans cluster, a linear regression, dimension reduction, etc.) performs.

Specifically, we will look at:

1. Comparing clusters to the ‘ground truth’ - the wine dataset
2. Cross validation of linear regression - the crime dataset
3. Investigating multidimensional scaling - the london borough dataset
4. Visualising the overlap in clustering results

23.3.1 Clustering and ground truth

Load in the wine dataset. Details of the dataset are [here]<https://archive.ics.uci.edu/ml/datasets/wine>).

```
import pandas as pd  
  
df = pd.read_csv('data/wine.csv')
```

Look at our data.

```
df.head()
```

	Class	label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Na
0	1		14.23	1.71	2.43	15.6	127	2.80	3.06	0.2

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Na
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.2
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.3
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.2
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.3

There is a column called Class label that gives us the ground truth. The wines come from three different cultivars. Knowing the actual grouping helps us to identify how well our methods can capture this ground truth.

Following our process above, we should first get a sense of our data.

```
df.describe()
```

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Na
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	1.938202	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.76	0.2
std	0.775035	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	3.49	0.3
min	1.000000	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	2.69	0.3
25%	1.000000	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	3.24	0.3
50%	2.000000	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	3.85	0.2
75%	3.000000	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.69	0.2
max	3.000000	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	2.76	0.2

No missing data. The scales of our features vary (e.g., Magnesium is in the 100s whereas Hue is in the low single digits).

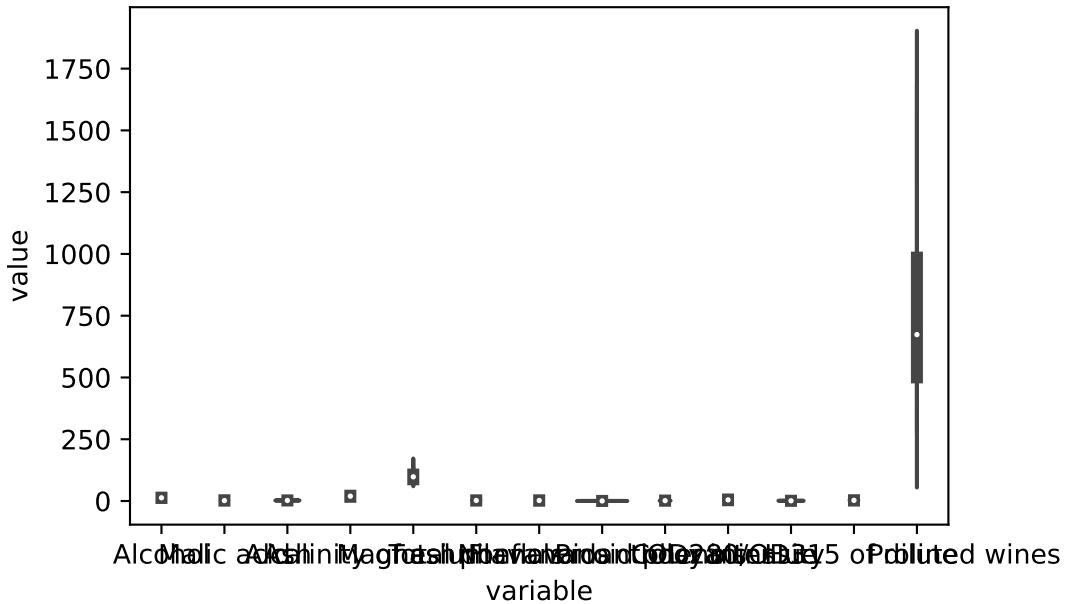
How about our feature distributions?

```
df_long = df.melt(id_vars='Class label')

import seaborn as sns

sns.violinplot(data = df_long, x = 'variable', y = 'value')

<Axes: xlabel='variable', ylabel='value'>
```



Makes sense to normalise our data.

```
from sklearn.preprocessing import MinMaxScaler

# create a scaler object
scaler = MinMaxScaler()

# fit and transform the data
df_norm = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)

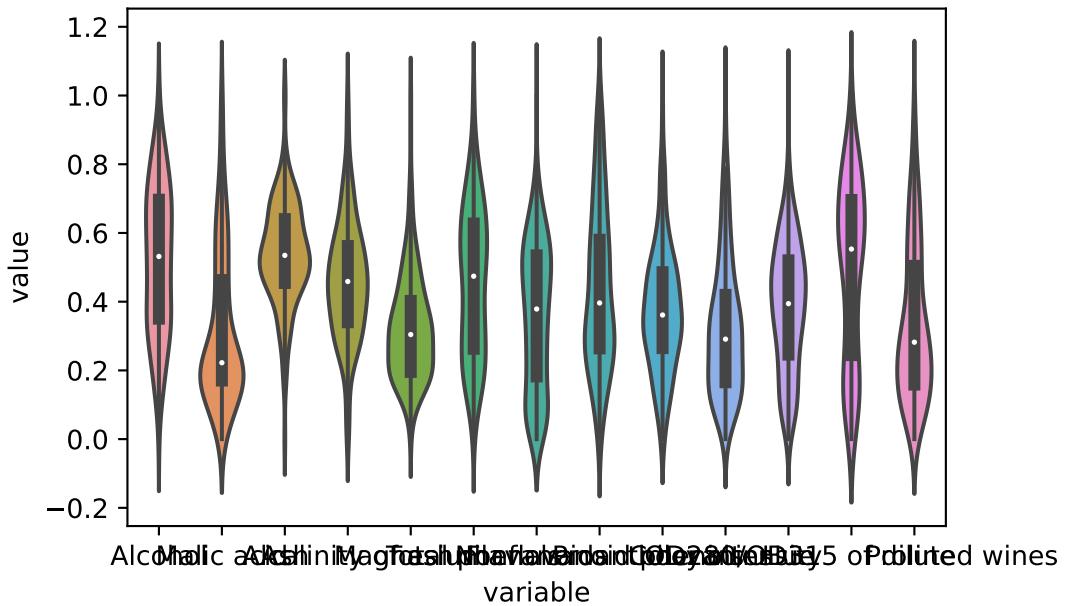
df_long = df_norm.melt(id_vars='Class label')
df_long
```

	Class label	variable	value
0	0.0	Alcohol	0.842105
1	0.0	Alcohol	0.571053
2	0.0	Alcohol	0.560526
3	0.0	Alcohol	0.878947
4	0.0	Alcohol	0.581579
...
2309	1.0	Proline	0.329529
2310	1.0	Proline	0.336662

	Class label	variable	value
2311	1.0	Proline	0.397290
2312	1.0	Proline	0.400856
2313	1.0	Proline	0.201141

```
sns.violinplot(data = df_long, x = 'variable', y = 'value')
```

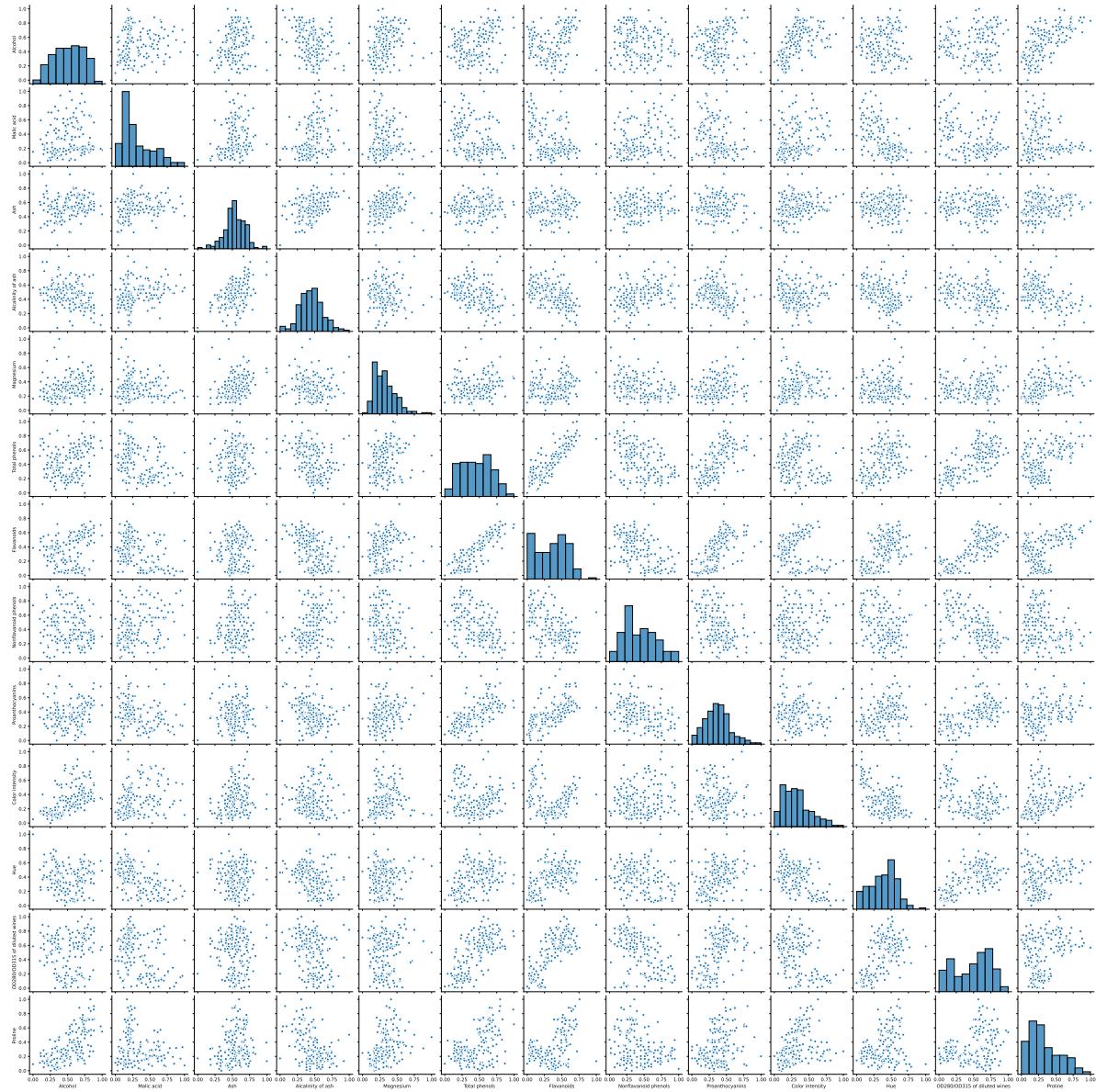
```
<Axes: xlabel='variable', ylabel='value'>
```



Are there any patterns?

How about a pairplot?

```
sns.pairplot(data = df_norm.iloc[:,1:])
```



Hmm, a few interesting correlations. Some of our variables are skewed. We could apply some PCA here to look at fewer dimension or even log transform some of the skewed variables.

For now we will just run a kmeans cluster and then check our results against the ground truth.

Lets decide how many clusters we need.

```

from sklearn.cluster import KMeans

ks = range(1, 10)
inertias = []
for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k, n_init = 10)

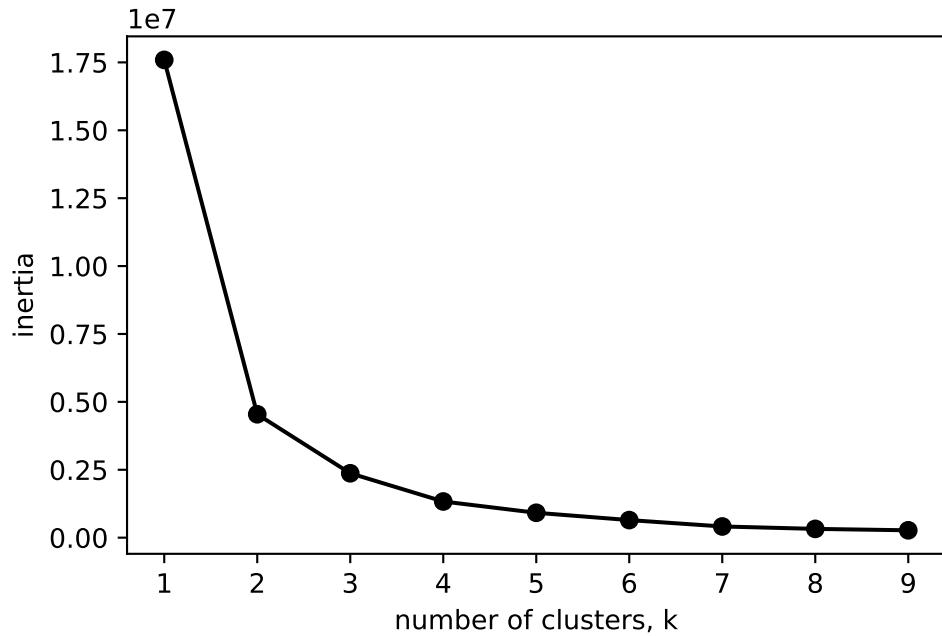
    # Fit model to samples
    model.fit(df.iloc[:,1:])

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

import matplotlib.pyplot as plt

plt.plot(ks, inertias, '-o', color='black')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()

```



What happens if we use the normalised data instead?

```
from sklearn.cluster import KMeans

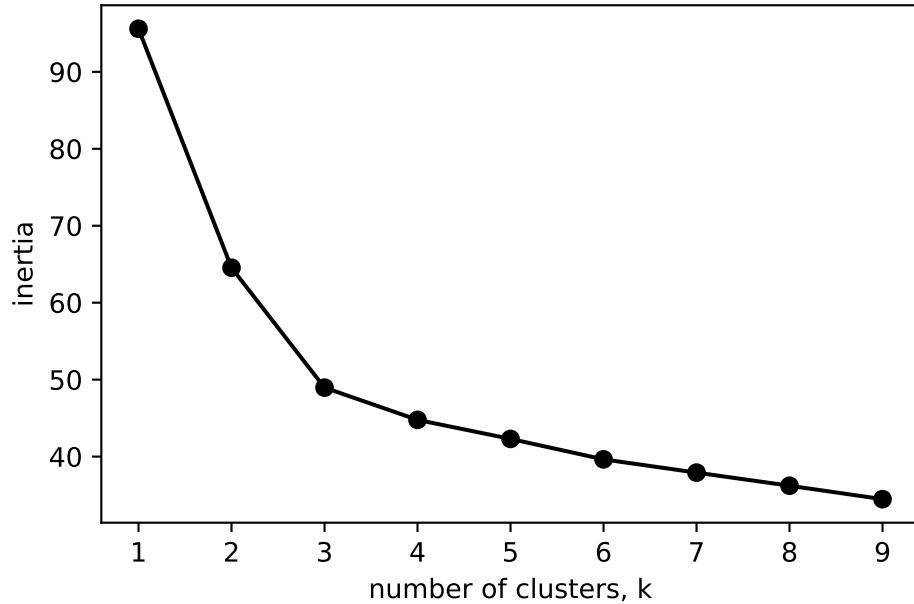
ks = range(1, 10)
inertias = []
for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k, n_init = 10)

    # Fit model to samples
    model.fit(df_norm.iloc[:,1:])

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

import matplotlib.pyplot as plt

plt.plot(ks, inertias, '-o', color='black')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```



Both of the graphs are the same. Is that what you would expect?

Three clusters seems about right (and matches our number of original labels).

```
df['Class label'].value_counts()
```

```
Class label
2    71
1    59
3    48
Name: count, dtype: int64
```

```
# Create a KMeans instance with k clusters: model
k_means = KMeans(n_clusters=3)

# Fit model to samples
df_k_means = k_means.fit(df.iloc[:,1:])

df['Three clusters'] = pd.Series(df_k_means.predict(df.iloc[:,1:]), index = df.index)
df
```

```
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/sklearn/cluster/_kmeans.py
    super().__check_params_vs_input(X, default_n_init=10)
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/sklearn/base.py:464: UserWarning:
  warnings.warn(
```

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69
...
173	3	13.71	5.65	2.45	20.5	95	1.68	0.61
174	3	13.40	3.91	2.48	23.0	102	1.80	0.75
175	3	13.27	4.28	2.26	20.0	120	1.59	0.69
176	3	13.17	2.59	2.37	20.0	120	1.65	0.68
177	3	14.13	4.10	2.74	24.5	96	2.05	0.76

Do our cluster labels match our ground truth? Did our cluster model capture reality?

```
ct = pd.crosstab(df['Three clusters'], df['Class label'])
ct
```

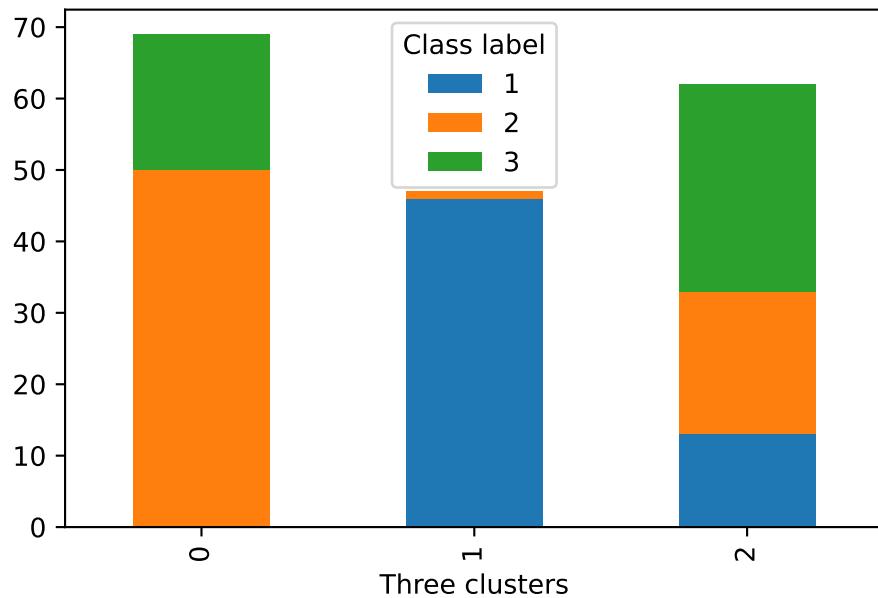
		Class label	1	2	3
		Three clusters			
0	0	0	50	19	
	1	46	1	0	
	2	13	20	29	

It might be easier to see as a stacked plot (see [this post](#)).

```
import matplotlib.pyplot as plt
import numpy as np

ct.plot.bar(stacked=True)
plt.legend(title='Class label')
```

```
<matplotlib.legend.Legend at 0x13fdd6310>
```



How has the kmeans model done compared to our ground truth?

We need to be really careful here. We notice that it is not easily possible to compare the known class labels to clustering labels. The reason is that the clustering algorithm labels are just arbitrary and not assigned to any deterministic criteria. Each time you run the algorithm, you might get a different id for the labels. The reason is that the label itself doesn't actually mean anything, what is important is the list of items that are in the same cluster and their relations.

A way to come over this ambiguity and evaluate the results is to look at a visualisations of the results and compare. But this brings in the question of what type of visualisation to use for looking at the clusters. An immediate alternative is to use scatterplots. However, it is not clear which axis to use for clustering. A common method to apply at this stage is to make use of PCA to get a 2D plane where we can project the data points and visualise them over this projection.

```
df.iloc[:,1:14]
```

	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39
...
173	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52
174	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43
175	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43
176	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53
177	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56

```
from sklearn.decomposition import PCA

n_components = 2

pca = PCA(n_components=n_components)
df_pca = pca.fit(df.iloc[:,1:14])
df_pca_vals = df_pca.transform(df.iloc[:,1:14])
```

Grab our projections and plot along with our cluster names.

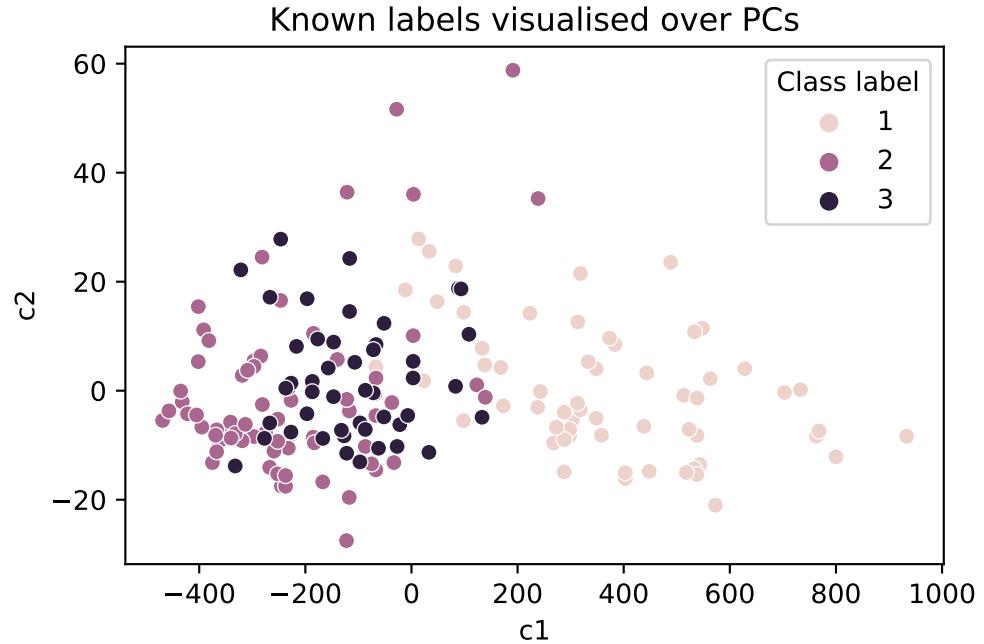
```
df['c1'] = [item[0] for item in df_pca_vals]
df['c2'] = [item[1] for item in df_pca_vals]
```

```

ax = sns.scatterplot(data = df, x = 'c1', y = 'c2', hue = 'Class label')
ax.set_title('Known labels visualised over PCs')

Text(0.5, 1.0, 'Known labels visualised over PCs')

```



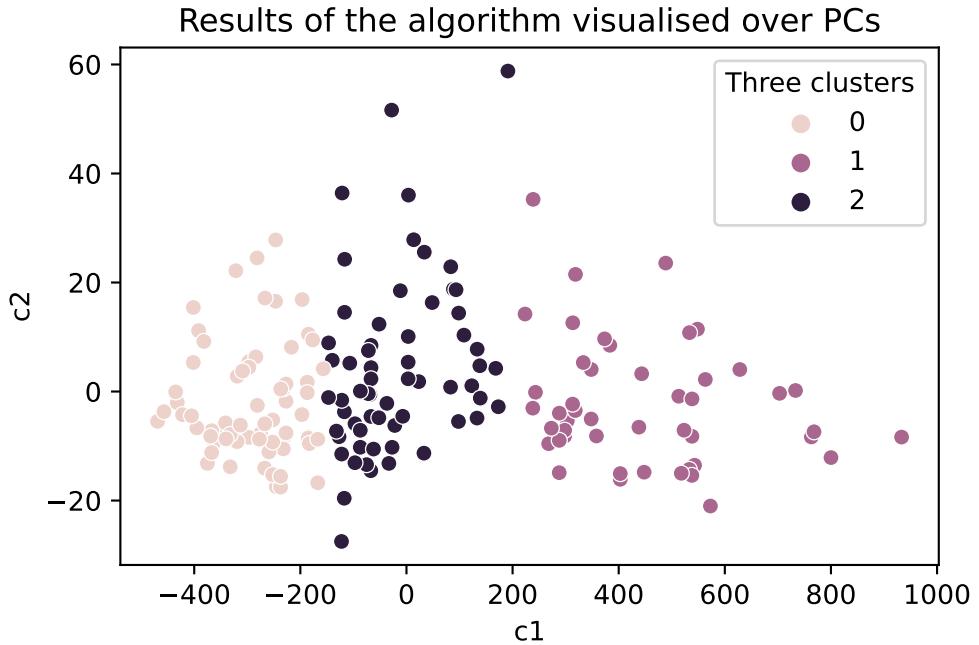
In the figure above, we colored the points based on the actual labels, we observe that there has been several misclassifications in the figure above (i.e., in the algorithm's results). So one may choose to use an alternative algorithm or devise a better distance metric.

```

ax = sns.scatterplot(data = df, x = 'c1', y = 'c2', hue = 'Three clusters')
ax.set_title('Results of the algorithm visualised over PCs')

Text(0.5, 1.0, 'Results of the algorithm visualised over PCs')

```



This shows the parallelism between the clustering algorithm and PCA. By looking at the PCA loadings, we can find out what the x-axis mean and try to interpret the clusters (We leave this as an additional exercise for those interested).

How might you interpret the above plots? Did the kmeans model identify the ground truth?

How robust is our clustering? It may be that the kmeans algorithm became stuck or that a few outliers have biased the clustering.

Two ways to check are:

- Running the model multiple times with different initial values.
- Removing some data and running the modelling multiple times.

Run the below cell a few times. What do you see?

```
# Create a KMeans instance with k clusters: model
k_means = KMeans(n_clusters=3, init='random', n_init = 10)

# Fit model to samples
df_k_means = k_means.fit(df.iloc[:,1:14])

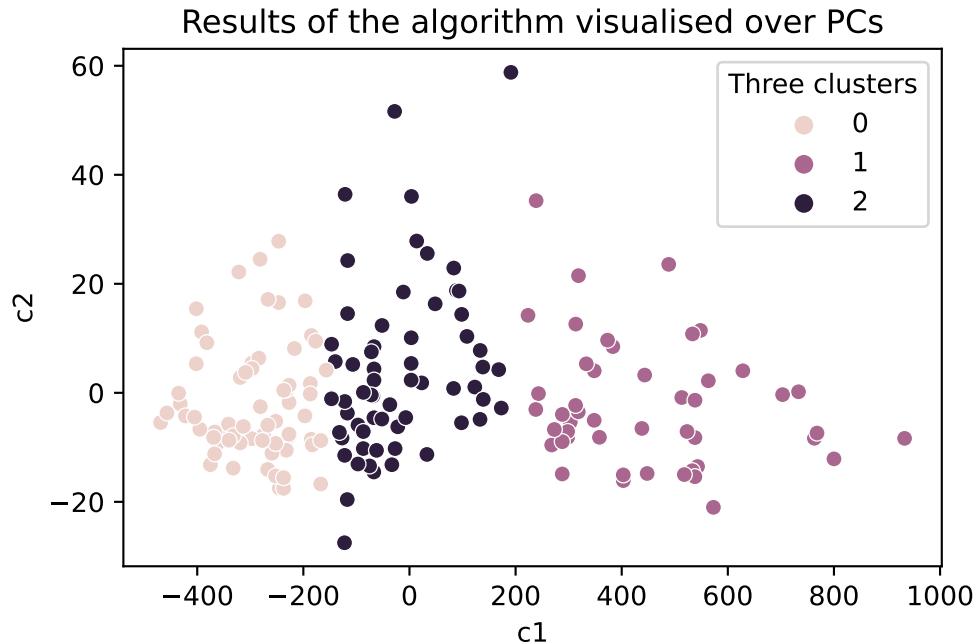
df['Three clusters'] = pd.Series(df_k_means.predict(df.iloc[:,1:14].values), index = df.in
```

```

ax = sns.scatterplot(data = df, x = 'c1', y = 'c2', hue = 'Three clusters')
ax.set_title('Results of the algorithm visualised over PCs')

/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/sklearn/base.py:464: UserWarning
  warnings.warn(
Text(0.5, 1.0, 'Results of the algorithm visualised over PCs')

```



How about with only 80% of the data?

```

df_sample = df.sample(frac=0.8, replace=False)

# Create a KMeans instance with k clusters: model
k_means = KMeans(n_clusters=3, init='random', n_init = 10)

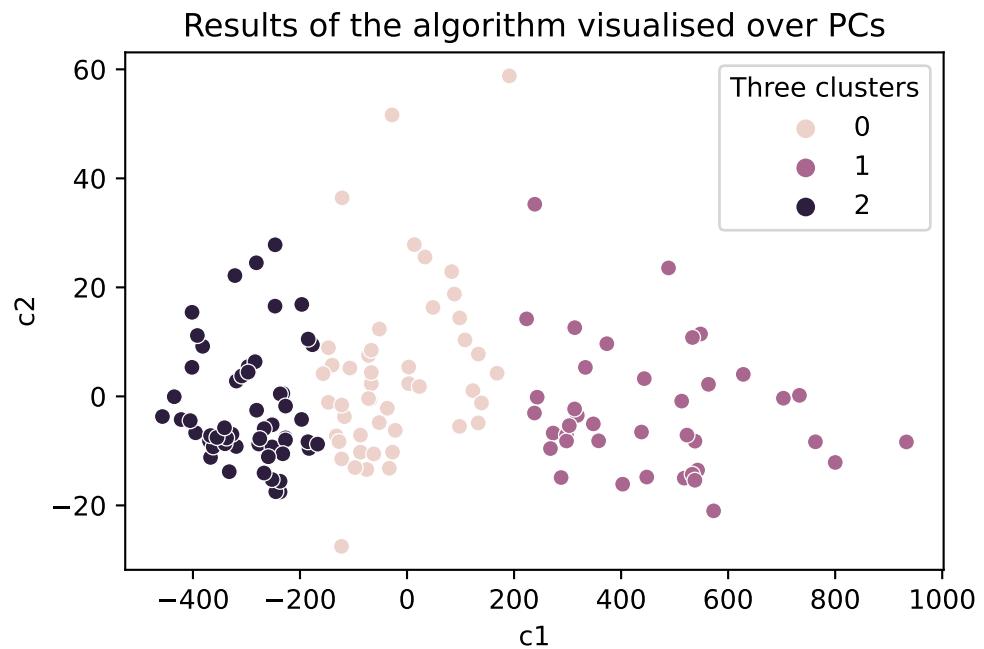
# Fit model to samples
df_k_means = k_means.fit(df_sample.iloc[:,1:14])

df_sample['Three clusters'] = pd.Series(df_k_means.predict(df_sample.iloc[:,1:14].values),

```

```
ax = sns.scatterplot(data = df_sample, x = 'c1', y = 'c2', hue = 'Three clusters')
ax.set_title('Results of the algorithm visualised over PCs')
```

```
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/sklearn/base.py:464: UserWarning
  warnings.warn(
Text(0.5, 1.0, 'Results of the algorithm visualised over PCs')
```



We may want to automate the process of resampling the data or rerunning the model then perhaps plotting the different inertia values or creating different plots.

Do you think our clustering algorithm is stable and provide similar results even when some data is removed or the initial values are random?

If so, then is our algorithm capturing the ground truth?

24 IM939 Lab 5 - Part 2

Details of the crime dataset are [here](#).

We are going to examine the data, fit and then cross-validate a regression model.

```
import pandas as pd
df = pd.read_csv('data/censusCrimeClean.csv')
df.head()
```

	communityname	fold	population	householdsize	racepctblack	racePctWhite	racePctAsian	ra
0	Lakewoodcity	1	0.19	0.33	0.02	0.90	0.12	0.
1	Tukwilacity	1	0.00	0.16	0.12	0.74	0.45	0.
2	Aberdeentown	1	0.00	0.42	0.49	0.56	0.17	0.
3	Willingborotownship	1	0.04	0.77	1.00	0.08	0.12	0.
4	Bethlehemtownship	1	0.01	0.55	0.02	0.95	0.09	0.

One hundred features. Too many for us to visualise at once.

Instead, we can pick out particular variables and carry out a linear regression. To make our work simple we will look at `ViolentCrimesPerPop` as our dependent variable and `medIncome` as our independent variable.

We may wonder if there is more violent crime in low income areas.

Let us create a new dataframe containing our regression variables. We do not have to do this I find it makes our work clearer.

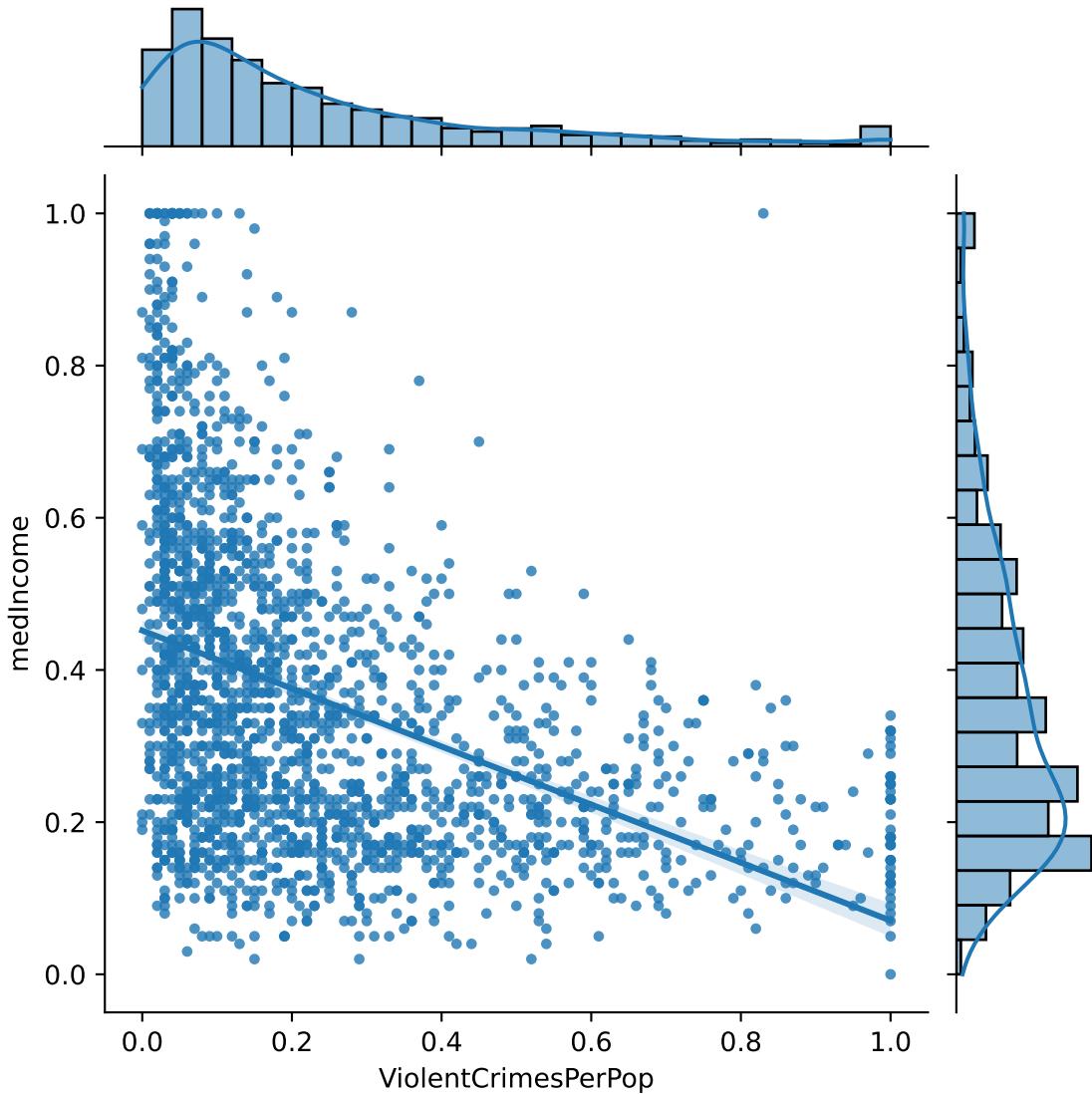
```
df_reg = df[['communityname', 'medIncome', 'ViolentCrimesPerPop']]
df_reg
```

	communityname	medIncome	ViolentCrimesPerPop
0	Lakewoodcity	0.37	0.20
1	Tukwilacity	0.31	0.67
2	Aberdeentown	0.30	0.43
3	Willingborotownship	0.58	0.12

	communityname	medIncome	ViolentCrimesPerPop
4	Bethlehemtownship	0.50	0.03
...
1989	TempleTerracecity	0.42	0.09
1990	Seasidecity	0.28	0.45
1991	Waterburytown	0.31	0.23
1992	Walthamcity	0.44	0.19
1993	Ontariocity	0.40	0.48

Plot our data (a nice page on plotting regressions with seaborn is [here](#)).

```
import seaborn as sns
sns.jointplot(data = df[['medIncome', 'ViolentCrimesPerPop']],
               x = 'ViolentCrimesPerPop',
               y = 'medIncome', kind='reg',
               marker = '.')
```



We may want to z-transform or log these scores as they are heavily skewed.

```
import numpy as np

# some values are 0 so 0.1 is added to prevent log giving us infinity
# there may be a better way to do this!
df_reg['ViolentCrimesPerPop_log'] = np.log(df_reg['ViolentCrimesPerPop'] + 0.1)
df_reg['medIncome_log'] = np.log(df_reg['medIncome'] + 0.1)
```

```
/var/folders/7v/zl9mv52s3ls94kntlt_19ryh0000gq/T/ipykernel_4508/1977719955.py:5: SettingWithCopyWarning
  A value is trying to be set on a copy of a slice from a DataFrame.
  Try using .loc[row_indexer,col_indexer] = value instead
```

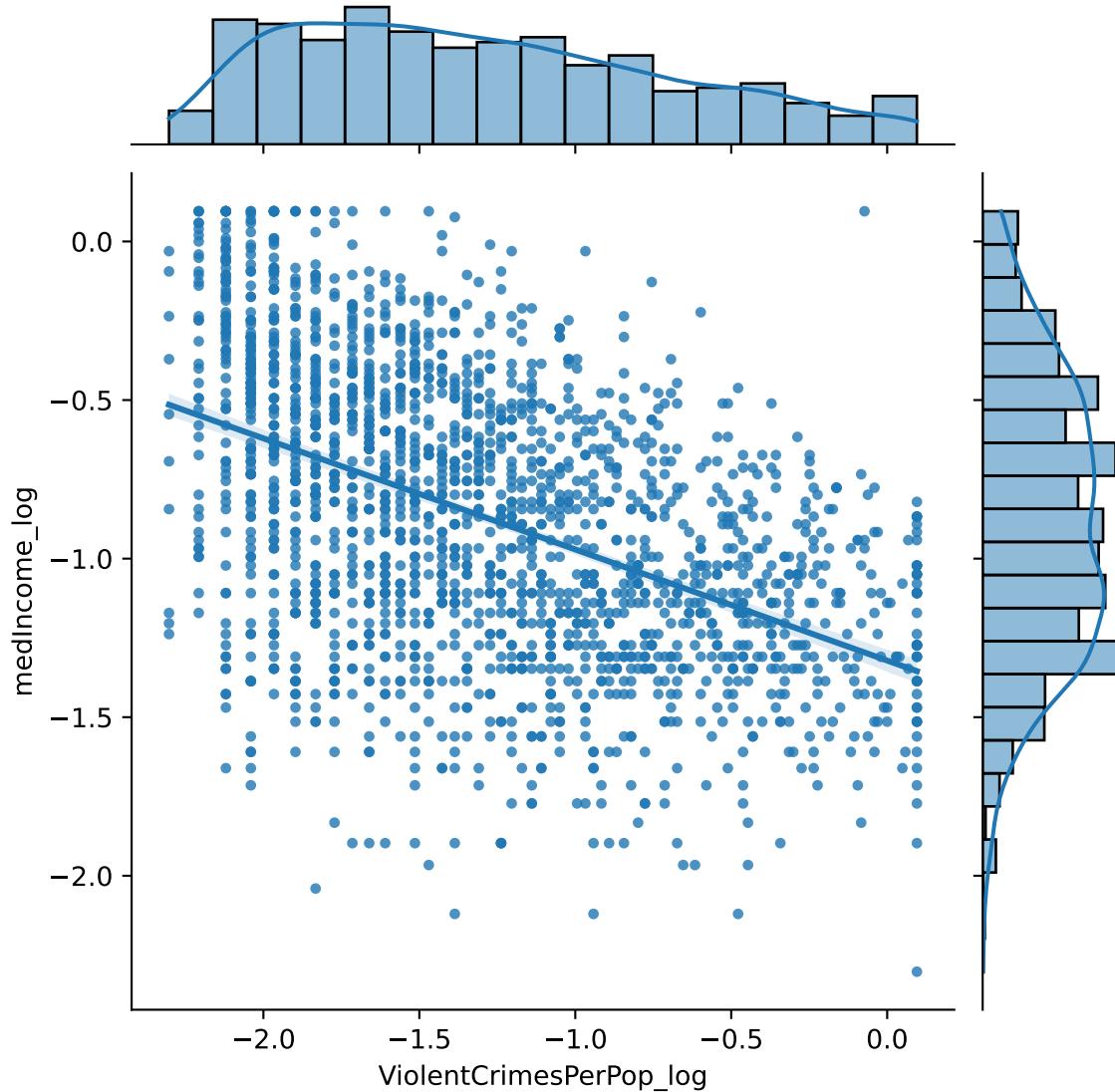
```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#inplace-mutation-vs-assignment
  df_reg['ViolentCrimesPerPop_log'] = np.log(df_reg['ViolentCrimesPerPop'] + 0.1)
/var/folders/7v/zl9mv52s3ls94kntlt_19ryh0000gq/T/ipykernel_4508/1977719955.py:6: SettingWithCopyWarning
  A value is trying to be set on a copy of a slice from a DataFrame.
  Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#inplace-mutation-vs-assignment
  df_reg['medIncome_log'] = np.log(df_reg['medIncome'] + 0.1)
```

```
df_reg
```

	communityname	medIncome	ViolentCrimesPerPop	ViolentCrimesPerPop_log	medIncome_log
0	Lakewoodcity	0.37	0.20	-1.203973	-0.755023
1	Tukwilacity	0.31	0.67	-0.261365	-0.891598
2	Aberdeentown	0.30	0.43	-0.634878	-0.916291
3	Willingborotownship	0.58	0.12	-1.514128	-0.385662
4	Bethlehemptownship	0.50	0.03	-2.040221	-0.510826
...
1989	TempleTerracecity	0.42	0.09	-1.660731	-0.653926
1990	Seasidecity	0.28	0.45	-0.597837	-0.967584
1991	Waterburytown	0.31	0.23	-1.108663	-0.891598
1992	Walthamcity	0.44	0.19	-1.237874	-0.616186
1993	Ontariocity	0.40	0.48	-0.544727	-0.693147

```
import seaborn as sns
sns.jointplot(data = df_reg[['medIncome_log', 'ViolentCrimesPerPop_log']],
               x = 'ViolentCrimesPerPop_log',
               y = 'medIncome_log', kind='reg',
               marker = '.')
```



Is log transforming our variables the right thing to do here?

Fit our regression to the log transformed data.

```

import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn import metrics

x = df_reg[['ViolentCrimesPerPop_log']]

```

```

y = df_reg[['medIncome_log']]

model = LinearRegression()
model.fit(x, y)

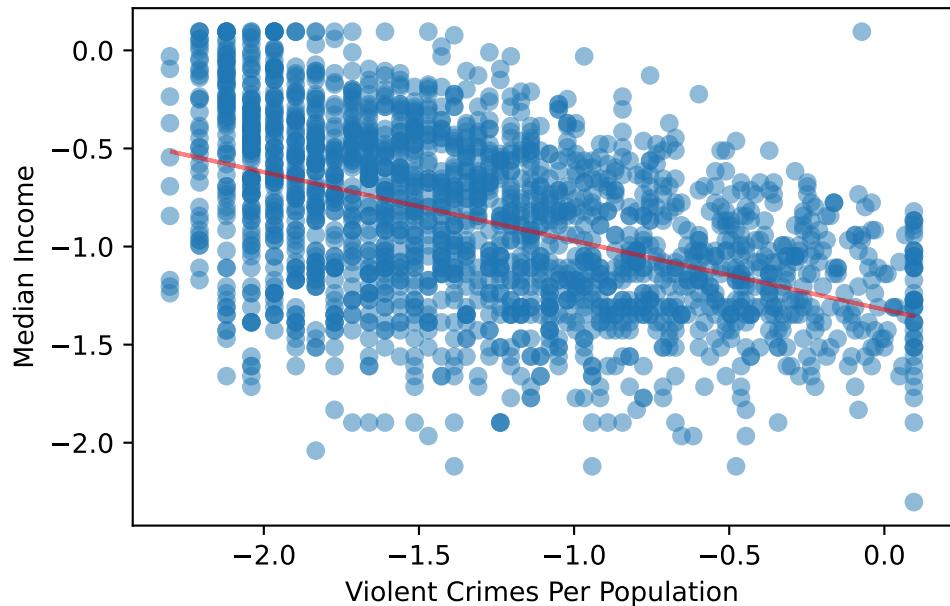
y_hat = model.predict(x)
plt.plot(x, y, 'o', alpha = 0.5)
plt.plot(x, y_hat, 'r', alpha = 0.5)

plt.xlabel('Violent Crimes Per Population')
plt.ylabel('Median Income')

print ("MSE:", metrics.mean_squared_error(y_hat, y))
print ("R^2:", metrics.r2_score(y, y_hat))
print ("var:", y.var())

```

MSE: 0.1531885348757034
 R²: 0.22763497704356928
 var: medIncome_log 0.198436
 dtype: float64



Has our log transformation distorted the pattern in the data?

```

x = df_reg[['ViolentCrimesPerPop']]
y = df_reg[['medIncome']]

model = LinearRegression()
model.fit(x, y)

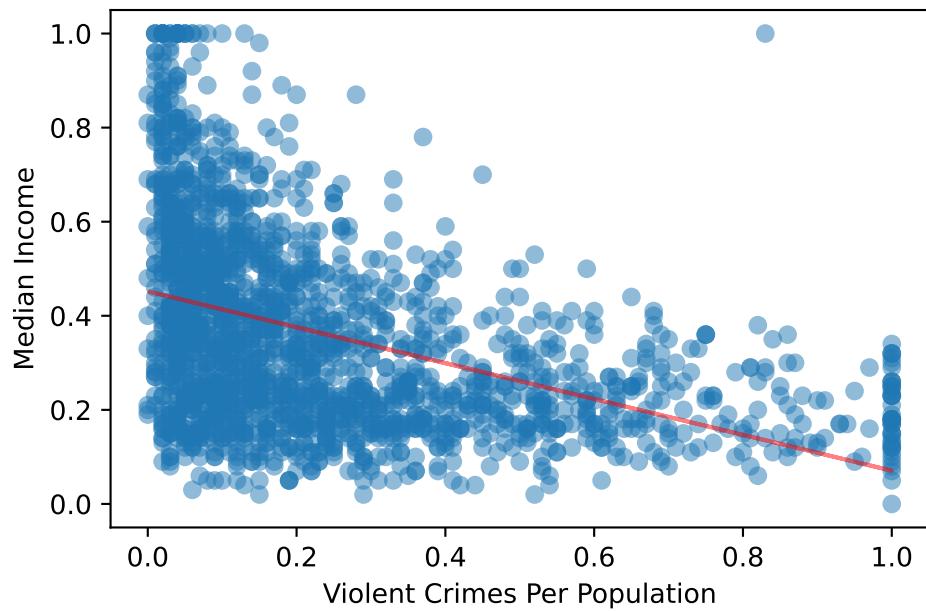
y_hat = model.predict(x)
plt.plot(x, y, 'o', alpha = 0.5)
plt.plot(x, y_hat, 'r', alpha = 0.5)

plt.xlabel('Violent Crimes Per Population')
plt.ylabel('Median Income')

print ("MSE:", metrics.mean_squared_error(y_hat, y))
print ("R^2:", metrics.r2_score(y, y_hat))
print ("var:", y.var())

```

MSE: 0.03592636778157073
R²: 0.17996313165549482
var: medIncome 0.043833
dtype: float64



What is the relationship between violent crime and median income? Why might this be?

Assuming the log data is fine, have we overfit the model? Remember that a good model (which accurately models the relationship between violent crimes per population) need to be robust when faced with new data.

Kfold cross validation splits data into train and test subsets. We can then fit the regression to the training set and see how well it does for the test set.

```
from sklearn.model_selection import KFold

X = df_reg[['ViolentCrimesPerPop']]
y = df_reg[['medIncome']]

# get four splits, Each split contains a
# test series and a train series.
kf = KFold(n_splits=4)

# lists to store our statistics
r_vals = []
MSEs = []
medIncome_coef = []

for train_index, test_index in kf.split(X):
    # fit our model and extract statistics
    model = LinearRegression()
    model.fit(X.iloc[train_index], y.iloc[train_index])
    y_hat = model.predict(X.iloc[test_index])

    MSEs.append(metrics.mean_squared_error(y.iloc[test_index], y_hat))
    medIncome_coef.append(model.coef_[0][0])
    r_vals.append(metrics.r2_score(y.iloc[test_index], y_hat))

data = {'MSE' : MSEs, 'medIncome coefficient' : medIncome_coef, 'r squared' : r_vals}
pd.DataFrame(data)
```

	MSE	medIncome coefficient	r squared
0	0.035727	-0.403609	0.130479
1	0.035904	-0.389344	0.162820
2	0.040777	-0.353379	0.200139
3	0.032255	-0.378883	0.182403

Does our model produce similar coefficients with subsets of the data?

We can do this using an inbuilt sklearn function (see [here](#)).

```
from sklearn.model_selection import cross_val_score
x = df_reg[['ViolentCrimesPerPop']]
y = df_reg[['medIncome']]

model = LinearRegression()
model.fit(x, y)

print(cross_val_score(model, x, y, cv=4))
```

```
[0.13047946 0.16281953 0.20013867 0.18240261]
```

What do these values tell us about our model and data?

You might want to carry out [multiple regression](#) with more than one predictor variable, or reduce the number of dimensions, or perhaps address different questions using a clustering algorithm instead with all or a subset of features.

25 IM939 Lab 5 - Part 3

Here we look at some London Borough data.

```
import pandas as pd

df = pd.read_excel('data/london-borough-profilesV3.xlsx', engine = 'openpyxl')
df.columns
```

Index(['Code', 'Area/INDICATOR', 'Inner/ Outer London',
 'GLA Population Estimate 2013', 'GLA Household Estimate 2013',
 'Inland Area (Hectares)', 'Population density (per hectare) 2013',
 'Average Age, 2013', 'Proportion of population aged 0-15, 2013',
 'Proportion of population of working-age, 2013',
 'Proportion of population aged 65 and over, 2013',
 '% of resident population born abroad (2013)',
 'Largest migrant population by country of birth (2013)',
 '% of largest migrant population (2013)',
 'Second largest migrant population by country of birth (2013)',
 '% of second largest migrant population (2013)',
 'Third largest migrant population by country of birth (2013)',
 '% of third largest migrant population (2013)',
 '% of population from BAME groups (2013)',
 '% people aged 3+ whose main language is not English (2011 census)',
 'Overseas nationals entering the UK (NINo), (2013/14)',
 'New migrant (NINo) rates, (2013/14)', 'Employment rate (%) (2013/14)',
 'Male employment rate (2013/14)', 'Female employment rate (2013/14)',
 'Unemployment rate (2013/14)', 'Youth Unemployment rate (2013/14)',
 'Proportion of 16-18 year olds who are NEET (%) (2013)',
 'Proportion of the working-age population who claim benefits (%) (Feb-2014)',
 '% working-age with a disability (2012)',
 'Proportion of working age people with no qualifications (%) 2013',
 'Proportion of working age people in London with degree or equivalent and above (%) 2013',
 'Gross Annual Pay, (2013)', 'Gross Annual Pay - Male (2013)',
 'Gross Annual Pay - Female (2013)',
 '% adults that volunteered in past 12 months (2010/11 to 2012/13)',

```
'Number of jobs by workplace (2012)',  
'% of employment that is in public sector (2012)', 'Jobs Density, 2012',  
'Number of active businesses, 2012',  
'Two-year business survival rates 2012',  
'Crime rates per thousand population 2013/14',  
'Fires per thousand population (2013)',  
'Ambulance incidents per hundred population (2013)',  
'Median House Price, 2013',  
'Average Band D Council Tax charge (£), 2014/15',  
'New Homes (net) 2012/13', 'Homes Owned outright, (2013) %',  
'Being bought with mortgage or loan, (2013) %',  
'Rented from Local Authority or Housing Association, (2013) %',  
'Rented from Private landlord, (2013) %',  
'% of area that is Greenspace, 2005', 'Total carbon emissions (2012)',  
'Household Waste Recycling Rate, 2012/13',  
'Number of cars, (2011 Census)',  
'Number of cars per household, (2011 Census)',  
'% of adults who cycle at least once per month, 2011/12',  
'Average Public Transport Accessibility score, 2012',  
'Indices of Multiple Deprivation 2010 Rank of Average Score',  
'Income Support claimant rate (Feb-14)',  
'% children living in out-of-work families (2013)',  
'Achievement of 5 or more A*- C grades at GCSE or equivalent including English and Maths',  
'Rates of Children Looked After (2013)',  
'% of pupils whose first language is not English (2014)',  
'Male life expectancy, (2010-12)', 'Female life expectancy, (2010-12)',  
'Teenage conception rate (2012)',  
'Life satisfaction score 2012-13 (out of 10)',  
'Worthwhileness score 2012-13 (out of 10)',  
'Happiness score 2012-13 (out of 10)',  
'Anxiety score 2012-13 (out of 10)', 'Political control in council',  
'Proportion of seats won by Conservatives in 2014 election',  
'Proportion of seats won by Labour in 2014 election',  
'Proportion of seats won by Lib Dems in 2014 election',  
'Turnout at 2014 local elections'],  
dtype='object')
```

```
df.head()
```

	Code	Area/INDICATOR	Inner/ Outer London	GLA Population Estimate 2013	GLA Hou
0	E09000001	City of London	Inner London	8000	4514.3713
1	E09000002	Barking and Dagenham	Outer London	195600	73261.408
2	E09000003	Barnet	Outer London	370000	141385.79
3	E09000004	Bexley	Outer London	236500	94701.226
4	E09000005	Brent	Outer London	320200	114318.55

Lots of different features. We also have really odd NaN values such as x and not available. We can try and get rid of this.

```
def isnumber(x):
    try:
        float(x)
        return True
    except:
        if (len(x) > 1) & ("not avail" not in x):
            return True
        else:
            return False

# apply isnumber function to every element
df = df[df.applymap(isnumber)]
df.head()
```

	Code	Area/INDICATOR	Inner/ Outer London	GLA Population Estimate 2013	GLA Hou
0	E09000001	City of London	Inner London	8000	4514.3713
1	E09000002	Barking and Dagenham	Outer London	195600	73261.408
2	E09000003	Barnet	Outer London	370000	141385.79
3	E09000004	Bexley	Outer London	236500	94701.226
4	E09000005	Brent	Outer London	320200	114318.55

That looks much cleaner.

Replace the NaN values in numeric columns with the mean.

```
# get only numeric columns
numericColumns = df._get_numeric_data()
```

```

from sklearn.metrics import euclidean_distances

# keep place names and store them in a variable
placeNames = df["Area/INDICATOR"]

# let's fill the missing values with mean()
numericColumns = numericColumns.fillna(numericColumns.mean())

# let's centralize the data
numericColumns -= numericColumns.mean()

# now we compute the euclidean distances between the columns by passing the same data twice
# the resulting data matrix now has the pairwise distances between the boroughs.
# CAUTION: note that we are now building a distance matrix in a high-dimensional data space
# remember the Curse of Dimensionality -- we need to be cautious with the distance values
distMatrix = euclidean_distances(numericColumns, numericColumns)

```

Check to make sure everything looks ok.

```
numericColumns.head()
```

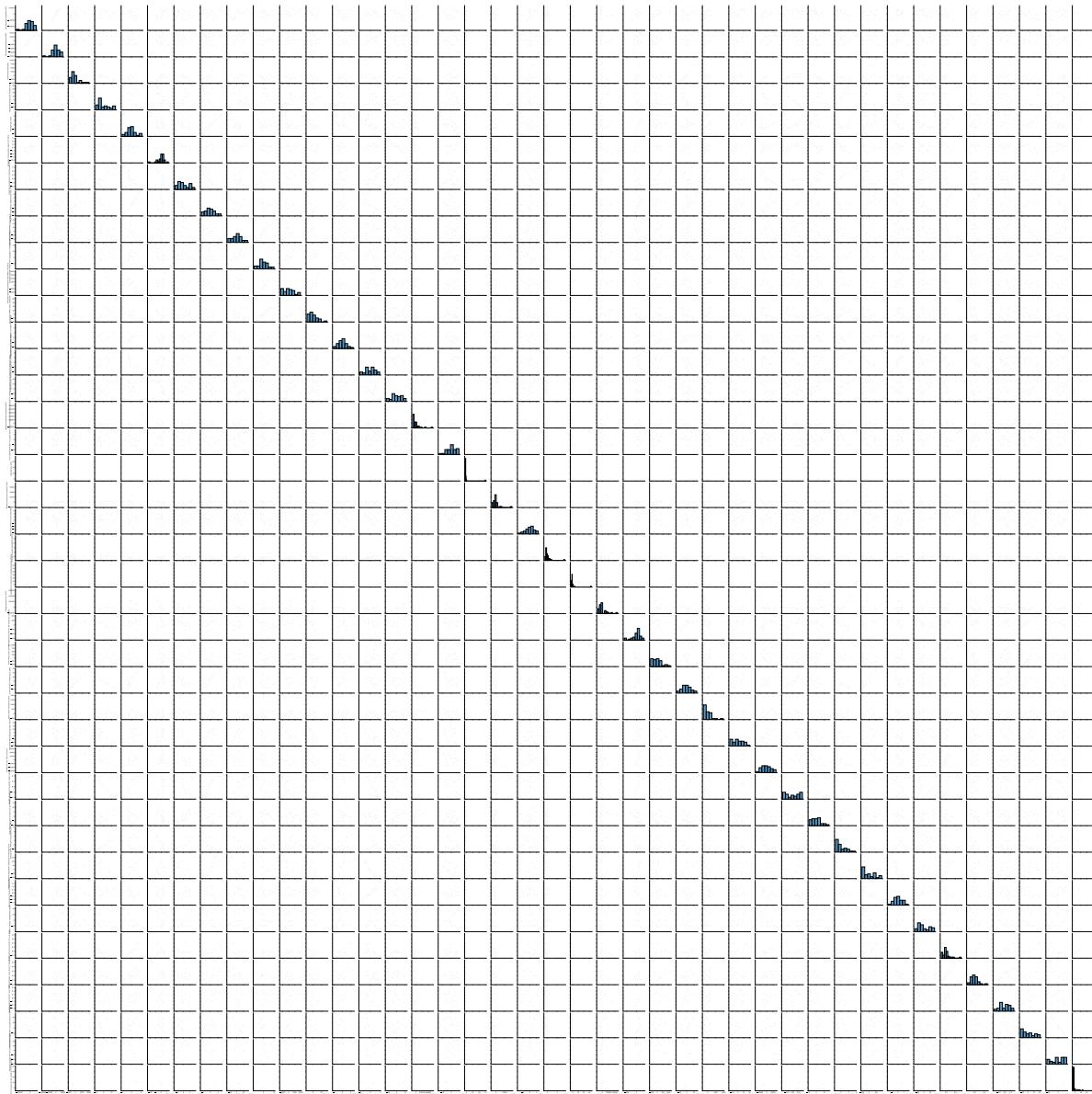
	GLA Population Estimate 2013	GLA Household Estimate 2013	Inland Area (Hectares)	Population density
0	-247760.606061	-97761.616805	-4473.681818	-43.279630
1	-60160.606061	-29014.579608	-1153.281818	-16.644971
2	114239.393939	39109.806712	3910.718182	-28.154125
3	-19260.606061	-7574.761788	1294.018182	-31.761255
4	64439.393939	12042.565712	-440.781818	3.258171

We can plot out our many dimension space. This will take quite a while (around 10 minutes).

```

import seaborn as sns
sns_plot = sns.pairplot(numericColumns)
sns_plot.savefig("output.png")

```



Dimension reduction will help us here!

We could apply various different types of dimension reduction here. We are specifically going to capture the dissimilarity in the data using [multidimensional scaling](#). Our distance matrix will come in useful here.

```
from sklearn import manifold
# for instance, typing distMatrix.shape on the console gives:
```

```

# Out[115]: (38, 38) # i.e., the number of rows

# first we generate an MDS object and extract the projections
mds = manifold.MDS(n_components = 2, max_iter=3000, n_init=1, dissimilarity="precomputed",
Y = mds.fit_transform(distMatrix)

```

To interpret what is happening, let us plot the boroughs on the projected two dimensional space.

```

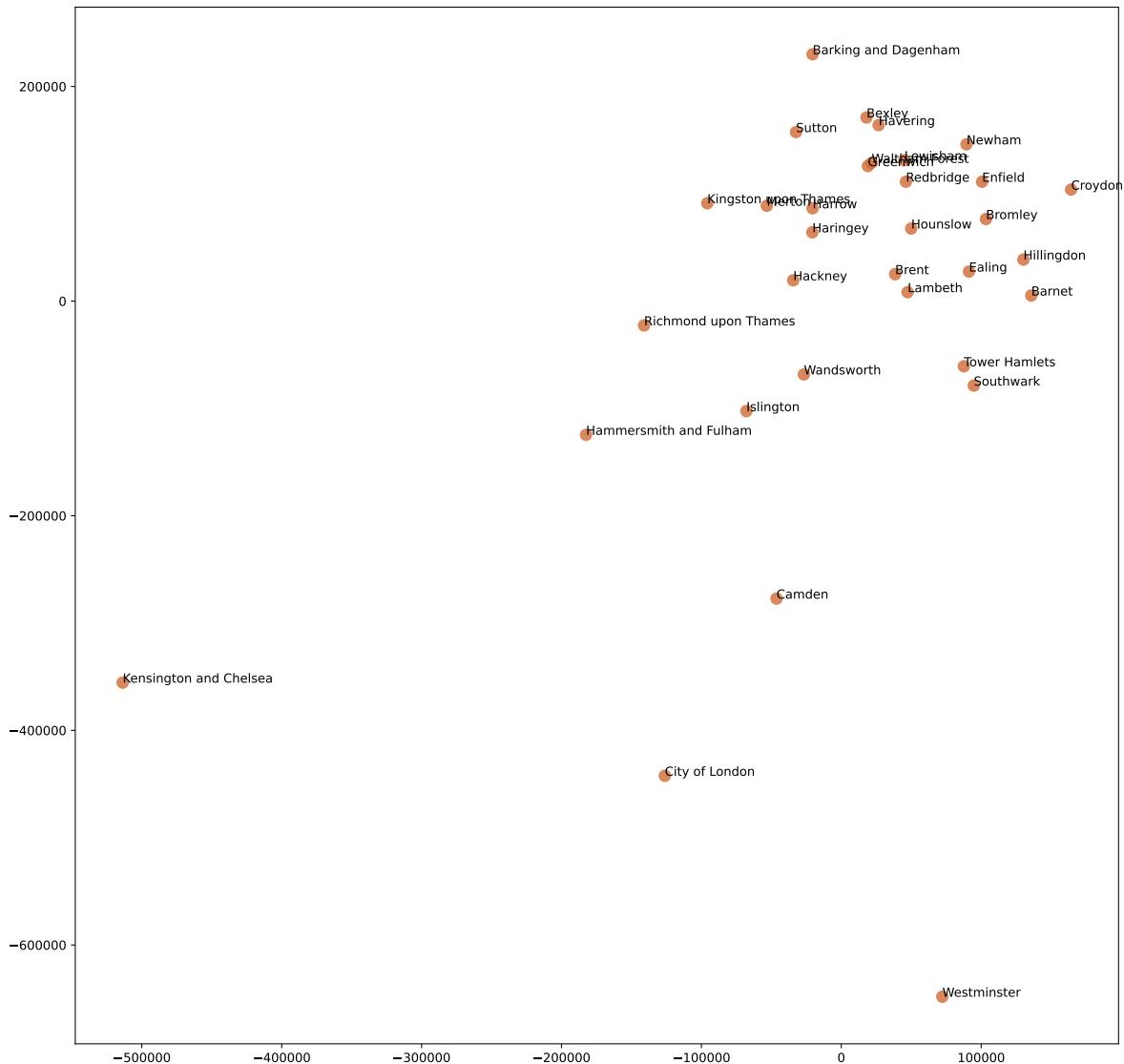
from matplotlib import pyplot as plt

fig, ax = plt.subplots()
fig.set_size_inches(15, 15)
plt.suptitle('MDS on only London boroughs')
ax.scatter(Y[:, 0], Y[:, 1], c="#D06B36", s = 100, alpha = 0.8, linewidth=0)

for i, txt in enumerate(placeNames):
    ax.annotate(txt, (Y[:, 0][i],Y[:, 1][i]))

```

MDS on only London boroughs



Our data also include happiness metrics. Pulling these out of our data and carrying out more multidimensional scaling can help us see how the boroughs differ in happiness.

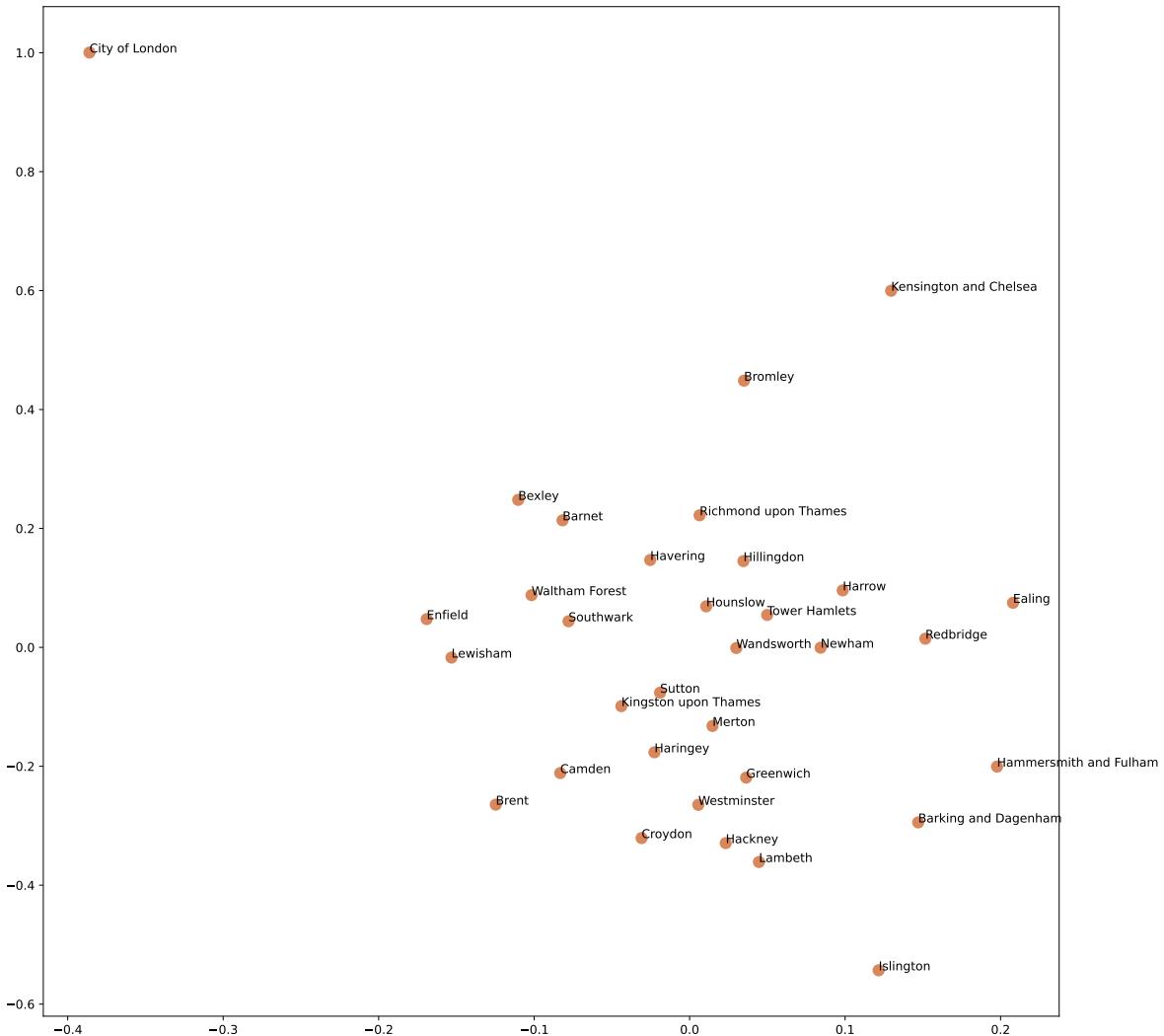
```
# get the data columns relating to emotions and feelings
dataOnEmotions = numericColumns[["Life satisfaction score 2012-13 (out of 10)", "Worthwhile",
```

```
# a new distance matrix to represent "emotional distance"s
distMatrix2 = euclidean_distances(dataOnEmotions, dataOnEmotions)

# compute a new "embedding" (machine learners' word for projection)
Y2 = mds.fit_transform(distMatrix2)

# let's look at the results
fig, ax = plt.subplots()
fig.set_size_inches(15, 15)
plt.suptitle('An \"emotional\" look at London boroughs')
ax.scatter(Y2[:, 0], Y2[:, 1], c="#D06B36", s = 100, alpha = 0.8, linewidth=0)

for i, txt in enumerate(placeNames):
    ax.annotate(txt, (Y2[:, 0][i],Y2[:, 1][i]))
```



The location of the different boroughs on the 2 dimensional multidimensional scaling space from the happiness metrics is

```
results_fixed = Y2.copy()
print(results_fixed)
```

```
[[ -3.85990240e-01  1.00035188e+00]
```

```
[ 1.46991846e-01 -2.94433493e-01]
[-8.17256802e-02  2.13639513e-01]
[-1.10243138e-01  2.48007826e-01]
[-1.24659381e-01 -2.64561645e-01]
[ 3.49846078e-02  4.48461738e-01]
[-8.32049455e-02 -2.11413672e-01]
[-3.08734897e-02 -3.20764747e-01]
[ 2.07998454e-01  7.48887373e-02]
[-1.69081765e-01  4.74918862e-02]
[ 3.63998563e-02 -2.19074578e-01]
[ 2.32446723e-02 -3.29253097e-01]
[ 1.97713361e-01 -2.00555398e-01]
[-2.26224006e-02 -1.76553318e-01]
[ 9.84626263e-02  9.57891313e-02]
[-2.53390298e-02  1.46913199e-01]
[ 3.46022361e-02  1.45001686e-01]
[ 1.06654654e-02  6.89338663e-02]
[ 1.21587739e-01 -5.43153065e-01]
[ 1.29598687e-01  5.99734425e-01]
[-4.38989108e-02 -9.88905500e-02]
[ 4.45786057e-02 -3.61016551e-01]
[-1.53167656e-01 -1.72181474e-02]
[ 1.47207519e-02 -1.32206971e-01]
[ 8.42493448e-02 -5.68831791e-04]
[ 1.51625001e-01  1.45681166e-02]
[ 6.41761708e-03  2.22082672e-01]
[-7.78651180e-02  4.38528215e-02]
[-1.89918894e-02 -7.62477759e-02]
[ 4.98885039e-02  5.44897041e-02]
[-1.01677894e-01  8.77594589e-02]
[ 3.00142708e-02 -1.18286819e-03]
[ 5.59789197e-03 -2.64871954e-01]]
```

We may want to look at if the general happiness rating captures the position of the boroughs. To do this we need to assign colours based on the binned happiness score.

```
import numpy as np

colorMappingValuesHappiness = np.asarray(dataOnEmotions[["Life satisfaction score 2012-13
print(results_fixed.shape)
colorMappingValuesHappiness.shape
```

```

colorMappingValuesHappiness
#c = colorMappingValuesCrime, cmap = plt.cm.Greens

(33, 2)

array([ 0.81636364, -0.22363636,  0.06636364,  0.18636364, -0.05363636,
       0.34636364, -0.06363636, -0.28363636, -0.04363636, -0.10363636,
      -0.12363636, -0.21363636, -0.05363636, -0.08363636,  0.05636364,
       0.11636364,  0.06636364,  0.01636364, -0.20363636,  0.39636364,
       0.00636364, -0.19363636, -0.05363636, -0.10363636, -0.06363636,
      -0.00363636,  0.13636364, -0.01363636, -0.03363636, -0.00363636,
      -0.04363636, -0.05363636, -0.19363636])

```

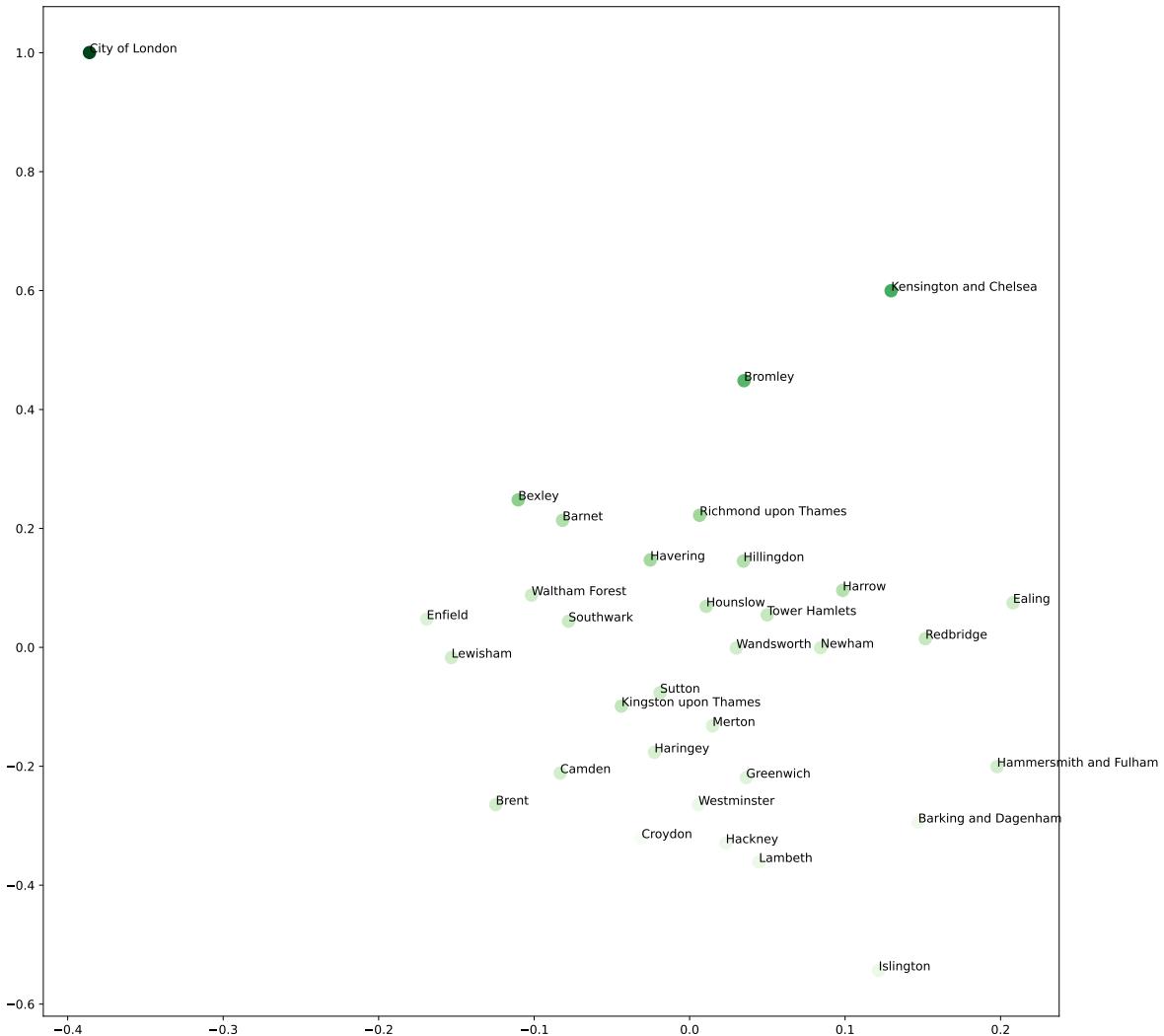
Finally, we can plot this. What can you see?

```

# let's look at the results
fig, ax = plt.subplots()
fig.set_size_inches(15, 15)
plt.suptitle('An \"emotional\" look at London boroughs')
#ax.scatter(results_fixed[:, 0], results_fixed[:, 1], c = colorMappingValuesHappiness, cmap = plt.cm.Greens)
plt.scatter(results_fixed[:, 0], results_fixed[:, 1], c = colorMappingValuesHappiness, s = 1000000)

for i, txt in enumerate(placeNames):
    ax.annotate(txt, (results_fixed[:, 0][i], results_fixed[:, 1][i]))

```



```

# get the data columns relating to emotions and feelings
dataOnDiversity = numericColumns[["Proportion of population aged 0-15, 2013", "Proportion
# a new distance matrix to represent "emotional distance"s
distMatrix3 = euclidean_distances(dataOnDiversity, dataOnDiversity)

mds = manifold.MDS(n_components = 2, max_iter=3000, n_init=1, dissimilarity="precomputed",

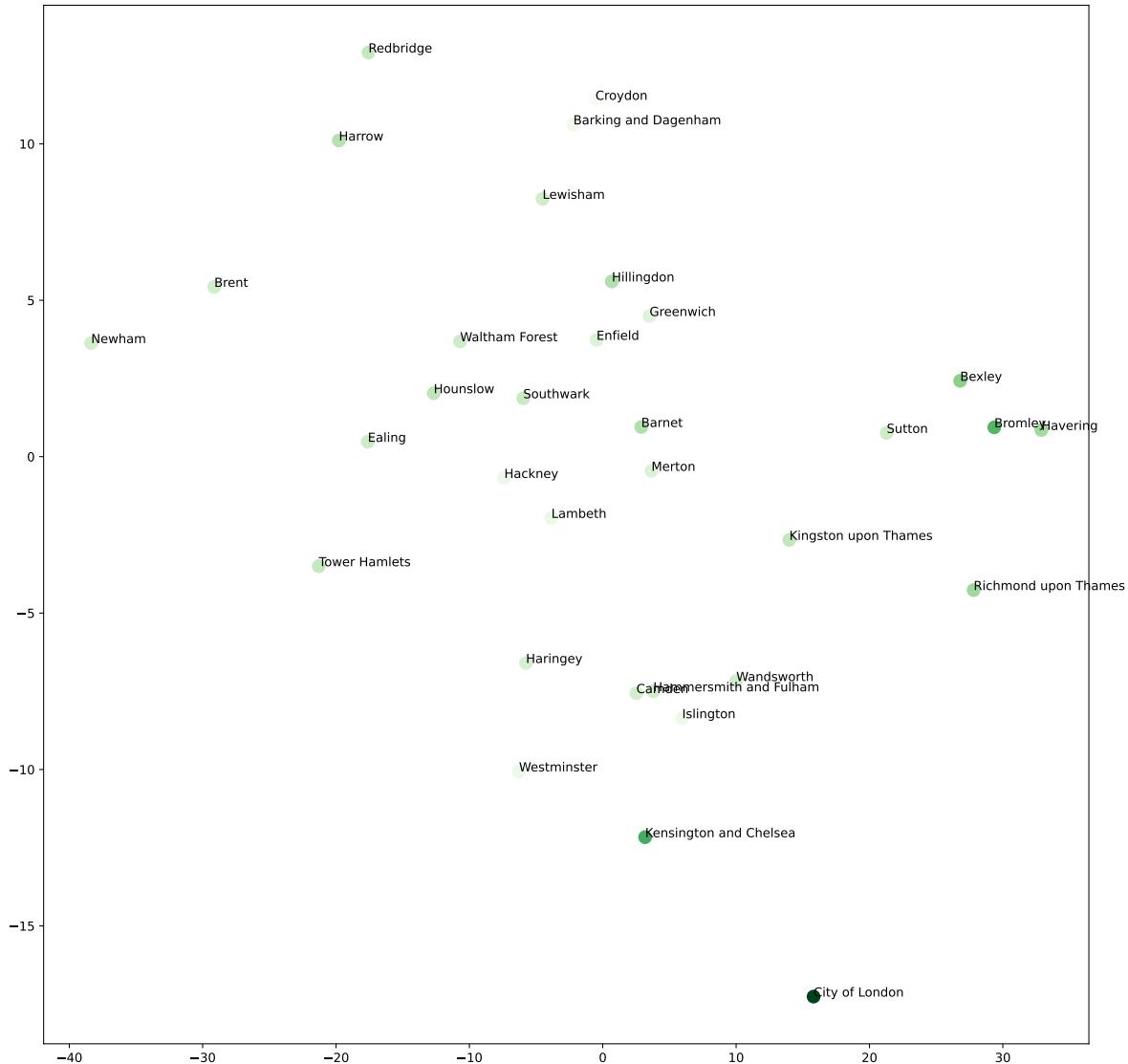
```

```
Y = mds.fit_transform(distMatrix3)

# Visualising the data.
fig, ax = plt.subplots()
fig.set_size_inches(15, 15)
plt.suptitle('An \"diversity\" look at London boroughs')
ax.scatter(Y[:, 0], Y[:, 1], s = 100, c = colorMappingValuesHappiness, cmap=plt.cm.Greens)

for i, txt in enumerate(placeNames):
    ax.annotate(txt, (Y[:, 0][i], Y[:, 1][i]))
```

An "diversity" look at London boroughs



This looks very different to the one above on “emotion” related variables. Our job now is to relate these two projections to one another. Do you see similarities? Do you see clusters of boroughs? Can you reflect on how you can relate and combine these two maps conceptually?

25.0.1 A small TODO for you:

Q: Can you think of other maps that you can produce with this data? Have a look at the variables once again and try to produce new “perspectives” to the data and see what they have to say.

26 IM939 lab 5 - Exercise

The exercise this week is a chance to apply the code you have been delving into over these past week.

Choose one of the datasets from a previous week:

- Crime Census
- London Borough
- Wine
- Iris
- Global warming

Or, if you are feeling confident, another dataset from the [sklearn datasets](#).

note Please do refer to the sklean and pandas documentation if you get stuck.

Read in the data using pandas.

Look at the first few rows. Get a feel for the structure of the data.

Deal with missing values, if any.

Create a summary of the data. Plot any particular features or groups of features which you think are of interest.

Settle on a possible question you want to answer. What might you be able to learn from your dataset?

Decide on your initial analysis. Remember, we have covered:

- Linear regressions
- Dimension reduction
- Clustering

Which method will best allow you to tackle your question?

Apply your chosen analysis method below. Please do refer to and copy and paste code from previous weeks.

Can you visualise your result?

You may want to use a:

- Scatterplot
- Histogram
- Any other plot, such as those in the [seaborn example library](#).

Are you able to check if your method is robust (e.g., kfold test of regressions or cluster stability checks)? Perhaps do that below.

Hmm, what have you learned?

You may want to consider if you could convince a friend of your conclusion. Perhaps another type of analysis is needed or there are issues with the analysis you chose above!

Use the space below to explore a bit more.

Part VII

Recognising and avoiding traps

27 IM939 - Lab 6 - Part 1 - Setup and Illusions

In this lab we are going to look at visualisations. We will consider 3 ‘illusions’ then misleading visualisations including bar plots, line plots and choropleths.

We will be using two new packages:

- Altair - a plotting library which has more flexibility than seaborn. It is a little more complex and examples can be found [here](#).
- Geopandas - an extension of the pandas library which allows us to work with geospatial data. The geopandas documentation is [here](#).

These packages are already included in the virtual environment for this module.

```
import altair as alt
import geopandas as gpd
```

27.1 Clustering illusion

You can find details of the clustering illusion [here](#), [here](#), and [here](#).

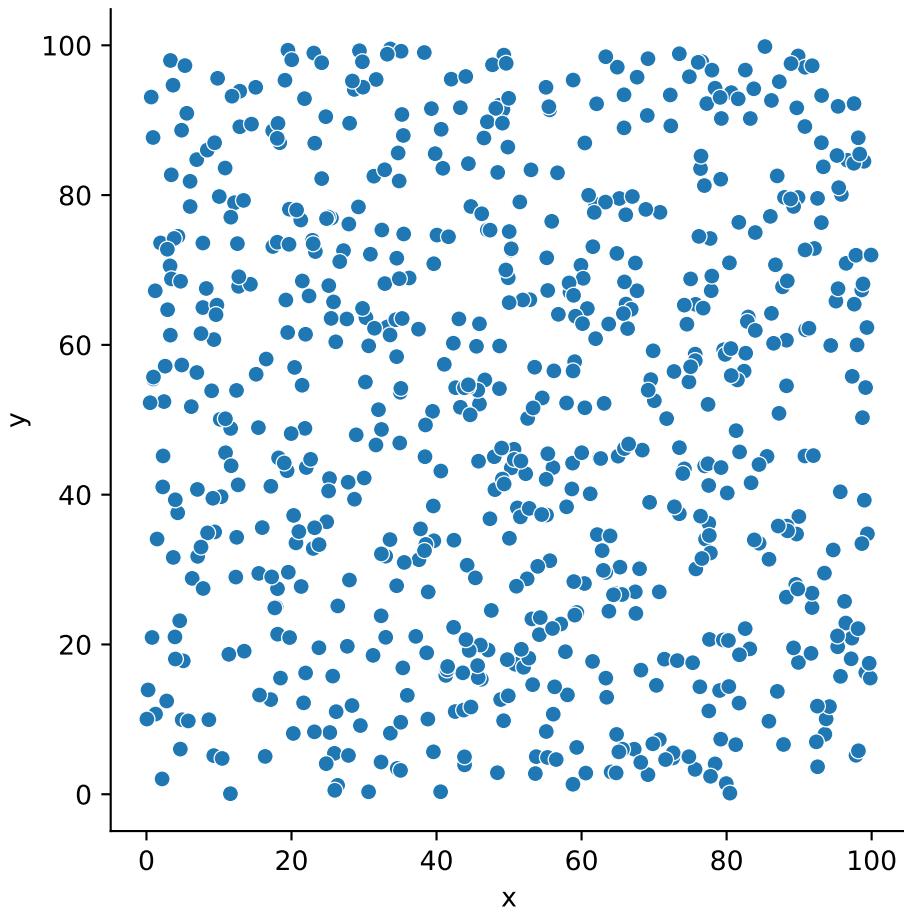
The illusion suggests you should automatically try and see clusters in random data.

```
import numpy as np
import seaborn as sns
import pandas as pd

n = 700

d = {'x': np.random.uniform(0, 100, n), 'y': np.random.uniform(0, 100, n)}
df = pd.DataFrame(d)

sns.relplot(data = df, x = 'x', y = 'y')
```



The syntax for building this type in Altair is pretty straight forward.

```
alt.Chart(df).mark_circle(size=5).encode(
    x='x',
    y='y')

alt.Chart(...)
```

To remove those pesky lines we need to specify we want an X and Y axis without grid lines.

```
alt.Chart(df).mark_circle(size=5).encode(
    alt.X('x', axis=alt.Axis(grid=False)),
    alt.Y('y', axis=alt.Axis(grid=False)))
```

```
alt.Chart(...)
```

We will do a lot of altering axis and colors in altair. We do this by specifying alt.axistype and then passing various options.

Do you see any clustering the in the above plots? What about if we give names to the different columns?

```
alt.Chart(df).mark_circle(size=5).encode(
    alt.X('x', axis=alt.Axis(grid=False, title='Height')),
    alt.Y('y', axis=alt.Axis(grid=False, title='Weight')))
```

```
alt.Chart(...)
```

Another example of the clustering illusion is the idea of ‘streaks’. That we see a pattern from a small sample and extrapolate out.

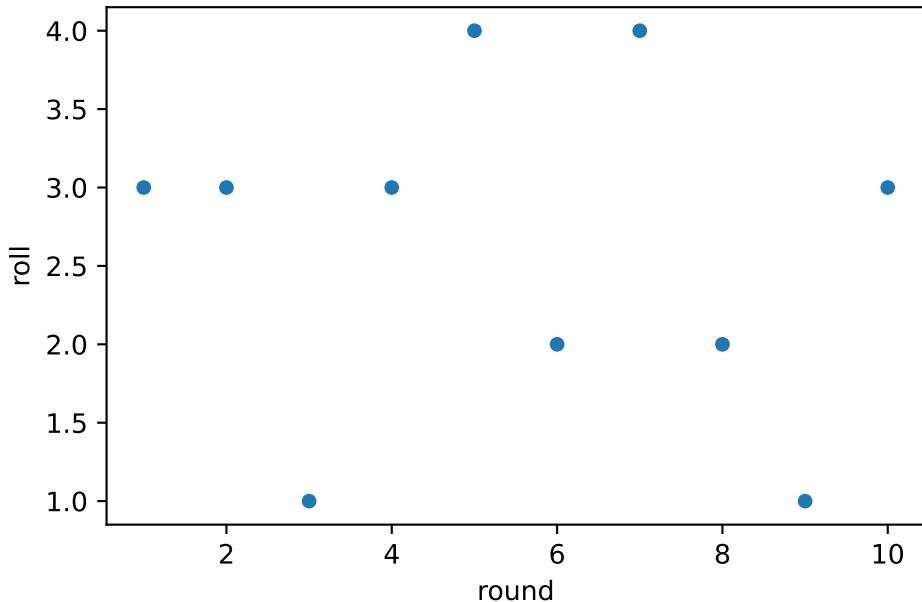
What do you expect the next dice role to be?

```
n_rolls = 10
d = {'round': np.linspace(1,n_rolls,n_rolls), 'roll': np.random.randint(1,6,n_rolls)}
df_dice = pd.DataFrame(d)
df_dice
```

	round	roll
0	1.0	3
1	2.0	3
2	3.0	1
3	4.0	3
4	5.0	4
5	6.0	2
6	7.0	4
7	8.0	2
8	9.0	1
9	10.0	3

```
sns.scatterplot(data=df_dice, x='round', y='roll')
```

```
<Axes: xlabel='round', ylabel='roll'>
```



```
alt.Chart(df_dice).mark_circle(size=20).encode(
    alt.X('round', axis=alt.Axis(grid=False)),
    alt.Y('roll', axis=alt.Axis(grid=False)))
```

```
alt.Chart(...)
```

Each number on the dice will occur the same number of times. Any patterns you see are due to extrapolating based on a small sample. We can check that though by rolling the 'dice' 1,000,000 times.

```
n_rolls = 1000000
d = {'round': np.linspace(1,n_rolls,n_rolls), 'roll': np.random.randint(1,6,n_rolls)}
df_dice_many = pd.DataFrame(d)

df_dice_many.groupby('roll').count()
```

roll	round
1	199906
2	200303
3	199695

roll	round
4	199895
5	200201

28 Weber-Fechner Law

Taken from Valdez, et. al. (2018):

'The Weber-Fechner Law is a famous finding of early psychophysics indicating that differences between stimuli are detected on a logarithmic scale. It takes more additional millimeters of radius to discern two larger circles than two smaller circles. This type of bias is probably one of the most researched biases in visualization research.'

Let us see if we can create a plot to demonstrate it.

We will load in the car crashes dataset from seaborn. Documentation of the data is [here](#).

```
df_crashes = sns.load_dataset('car_crashes')
df_crashes.head()
```

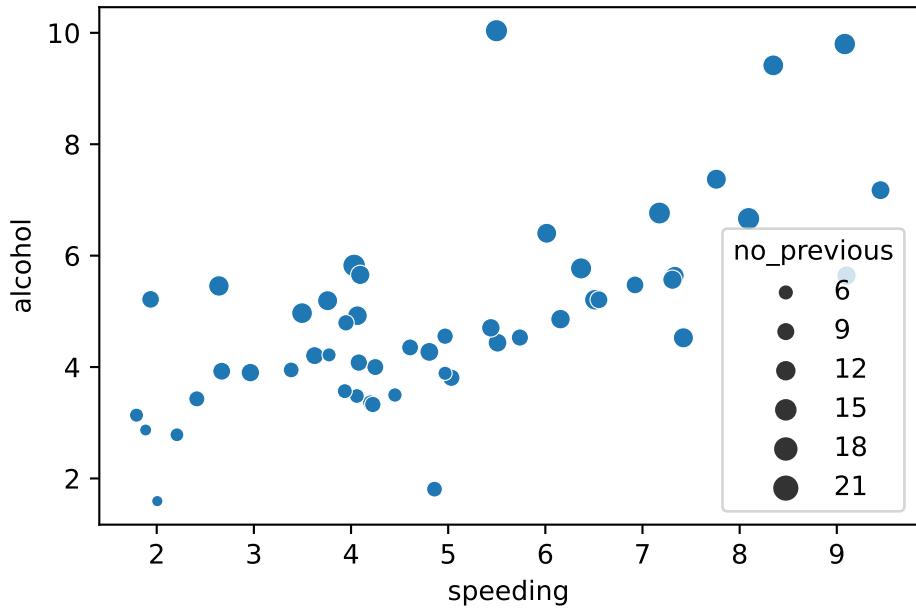
	total	speeding	alcohol	not_distracted	no_previous	ins_premium	ins_losses	abbrev
0	18.8	7.332	5.640	18.048	15.040	784.55	145.08	AL
1	18.1	7.421	4.525	16.290	17.014	1053.48	133.93	AK
2	18.6	6.510	5.208	15.624	17.856	899.47	110.35	AZ
3	22.4	4.032	5.824	21.056	21.280	827.34	142.39	AR
4	12.0	4.200	3.360	10.920	10.680	878.41	165.63	CA

To illustrate this 'illusion' we will plot the percentage of drivers speeding, percentage of alcohol impaired and set the size of the point equal to the percentage of drivers not previously involved in any accident. Each point is an american state.

Are there any relationships or patterns in the data?

```
sns.scatterplot(data=df_crashes, x='speeding', y='alcohol', size='no_previous')

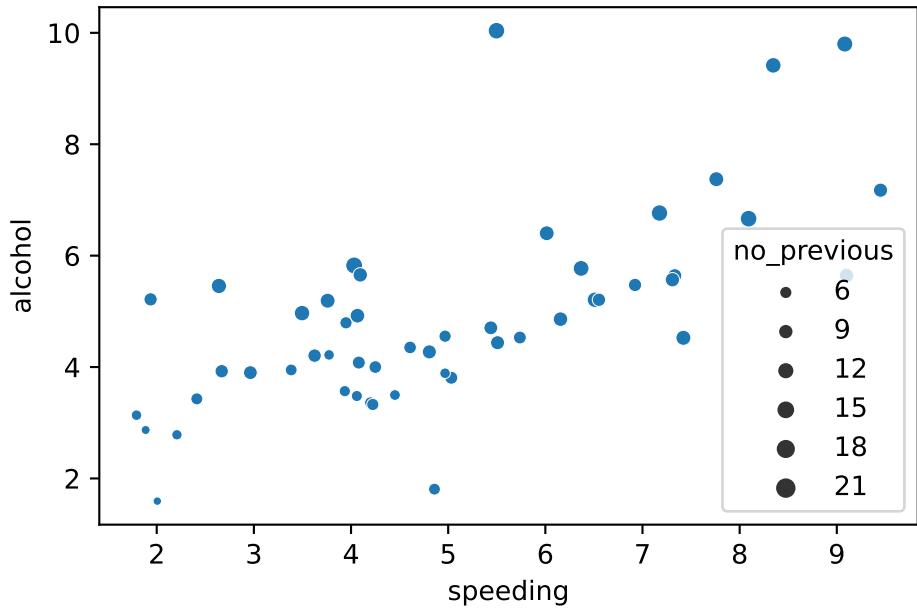
<Axes: xlabel='speeding', ylabel='alcohol'>
```



Is it easier to distinguish the different sizes in the below plot?

```
sns.scatterplot(data=df_crashes,
                 x='speeding',
                 y='alcohol',
                 size='no_previous',
                 sizes=(10, 40))
```

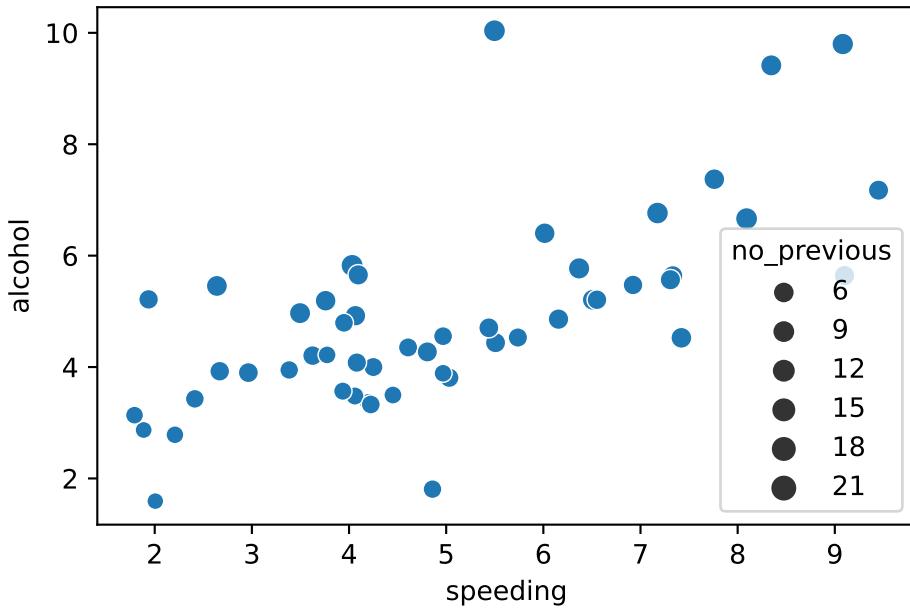
<Axes: xlabel='speeding', ylabel='alcohol'>



How about this one?

```
sns.scatterplot(data=df_crashes,
                 x='speeding',
                 y='alcohol',
                 size='no_previous',
                 sizes=(40,70))
```

<Axes: xlabel='speeding', ylabel='alcohol'>



The values are the same. We have just changed the range of sizes.

We can do much the same in altair.

```

alt.Chart(df_crashes).mark_circle().encode(
    x='speeding',
    y='alcohol',
    size='no_previous'
)

alt.Chart(...)

alt.Chart(df_crashes).mark_circle().encode(
    x='speeding',
    y='alcohol',
    size = alt.Size('no_previous', scale=alt.Scale(range=[10,40]))
)

alt.Chart(...)
```

```
alt.Chart(df_crashes).mark_circle().encode(
    x='speeding',
    y='alcohol',
    size = alt.Size('no_previous', scale=alt.Scale(range=[40,70]))
)

alt.Chart(...)
```

Have you come across any other illusions? If so, try and plot them out. I sometimes find it easier to understand these things through creating simple illustrations of my own.

29 IM939 - Lab 6 - Part 2 - Axes manipulation

One way to create potentially misleading visualisations is by manipulating the axes of a plot. Here we illustrate these using one of the FiveThirtyEight data sets, which are available [here](#).

29.1 Data wrangling

We are going to use polls from the recent USA presidential election. As before, we load and examine the data.

```
import pandas as pd
import seaborn as sns
import altair as alt

df_polls = pd.read_csv('data/presidential_poll_averages_2020.csv')
df_polls.head()
```

	cycle	state	modeldate	candidate_name	pct_estimate	pct_trend_adjusted
0	2020	Wyoming	11/3/2020	Joseph R. Biden Jr.	30.81486	30.82599
1	2020	Wisconsin	11/3/2020	Joseph R. Biden Jr.	52.12642	52.09584
2	2020	West Virginia	11/3/2020	Joseph R. Biden Jr.	33.49125	33.51517
3	2020	Washington	11/3/2020	Joseph R. Biden Jr.	59.34201	59.39408
4	2020	Virginia	11/3/2020	Joseph R. Biden Jr.	53.74120	53.72101

For our analysis, we are going to pick estimates from 11/3/2020 for the swing states of Florida, Texas, Arizona, Michigan, Minnesota and Pennsylvania.

```
df_nov = df_polls[
    (df_polls.modeldate == '11/3/2020')
]

df_nov = df_nov[
    (df_nov.candidate_name == 'Joseph R. Biden Jr.') |
```

```

(df_nov.candidate_name == 'Donald Trump')
]

df_swing = df_nov[
    (df_nov['state'] == 'Florida') |
    (df_nov['state'] == 'Texas') |
    (df_nov['state'] == 'Arizona') |
    (df_nov['state'] == 'Michigan') |
    (df_nov['state'] == 'Minnesota') |
    (df_nov['state'] == 'Pennsylvania')
]

df_swing

```

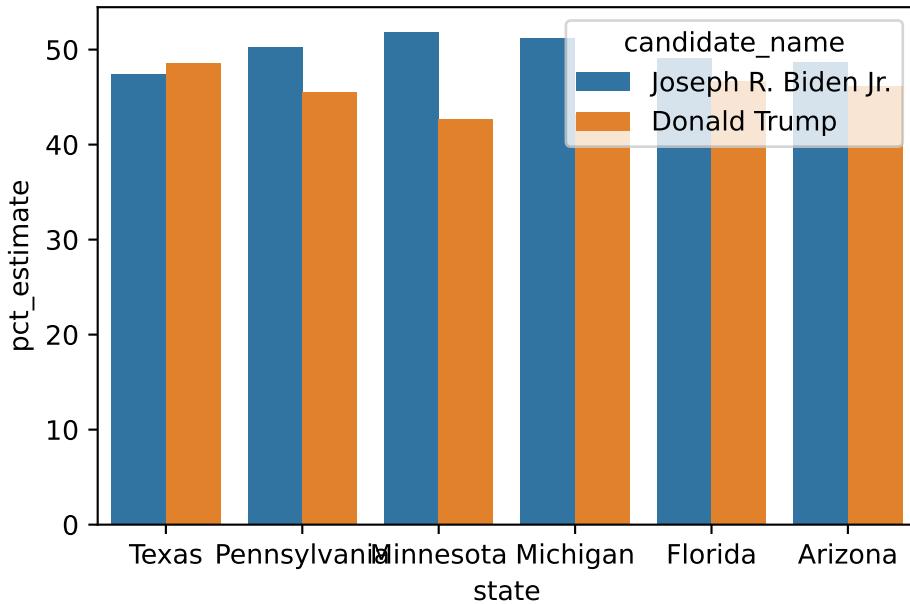
	cycle	state	modeldate	candidate_name	pct_estimate	pct_trend_adjusted
7	2020	Texas	11/3/2020	Joseph R. Biden Jr.	47.46643	47.44781
12	2020	Pennsylvania	11/3/2020	Joseph R. Biden Jr.	50.22000	50.20422
30	2020	Minnesota	11/3/2020	Joseph R. Biden Jr.	51.86992	51.84517
31	2020	Michigan	11/3/2020	Joseph R. Biden Jr.	51.17806	51.15482
46	2020	Florida	11/3/2020	Joseph R. Biden Jr.	49.09162	49.08035
53	2020	Arizona	11/3/2020	Joseph R. Biden Jr.	48.72237	48.70539
63	2020	Texas	11/3/2020	Donald Trump	48.57118	48.58794
68	2020	Pennsylvania	11/3/2020	Donald Trump	45.57216	45.55034
86	2020	Minnesota	11/3/2020	Donald Trump	42.63638	42.66826
87	2020	Michigan	11/3/2020	Donald Trump	43.20577	43.23326
102	2020	Florida	11/3/2020	Donald Trump	46.68101	46.61909
109	2020	Arizona	11/3/2020	Donald Trump	46.11074	46.10181

We can look at the relative performance of the candidates within each state using a nested bar plot.

```

ax = sns.barplot(
    data = df_swing,
    x = 'state',
    y = 'pct_estimate',
    hue = 'candidate_name')

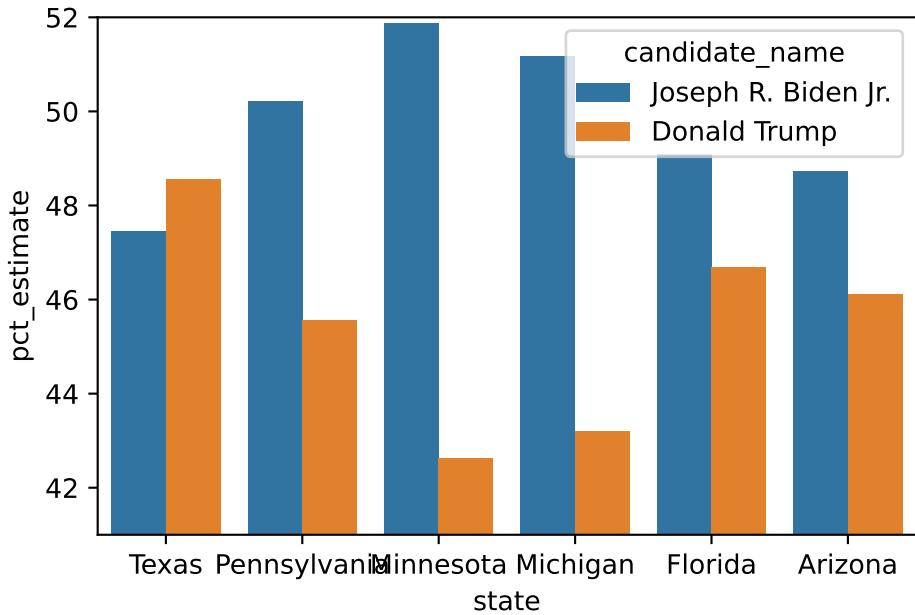
```



Altering the axis increases the distance between the bars. Some might say that is misleading.

```
ax = sns.barplot(  
    data = df_swing,  
    x = 'state',  
    y = 'pct_estimate',  
    hue = 'candidate_name')  
  
ax.set(ylim=(41, 52))
```

[(41.0, 52.0)]

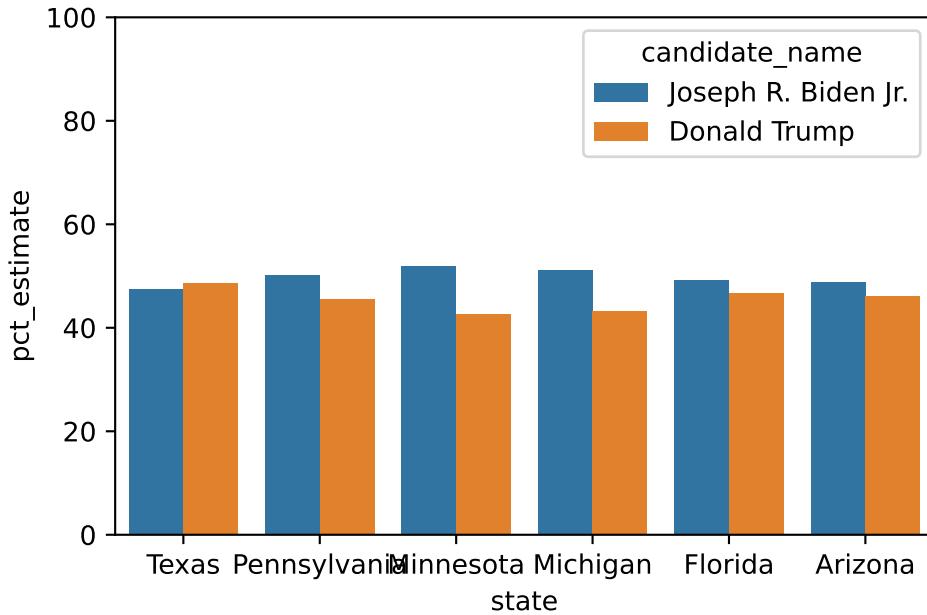


What do you think?

How about if we instead put the data on the full 0 to 100 scale?

```
ax = sns.barplot(  
    data = df_swing,  
    x = 'state',  
    y = 'pct_estimate',  
    hue = 'candidate_name')  
  
ax.set(ylim=(0, 100))
```

[(0.0, 100.0)]



We can do the same thing in Altair.

```
alt.Chart(df_swing).mark_bar().encode(
    x='candidate_name',
    y='pct_estimate',
    color='candidate_name',
    column = alt.Column('state:0', spacing = 5, header = alt.Header(labelOrient = "bottom"))
)

alt.Chart(...)
```

Note the need for the alt column. What happens if you do not provide an alt column?

Passing the domain option to the scale of the Y axis allows us to choose the y axis range.

```
alt.Chart(df_swing).mark_bar().encode(
    x='candidate_name',
    y=alt.Y('pct_estimate', scale=alt.Scale(domain=[42,53])),
    color='candidate_name',
    column = alt.Column('state:0', spacing = 5, header = alt.Header(labelOrient = "bottom"))
)
```

```
alt.Chart(...)
```

We can even be a bit tricky and stretch out the difference.

```
alt.Chart(df_swing).mark_bar().encode(
    x='candidate_name',
    y=alt.Y('pct_estimate', scale=alt.Scale(domain=[42,53])),
    color='candidate_name',
    column = alt.Column('state:0', spacing = 5, header = alt.Header(labelOrient = "bottom"))
).properties(
    width=20,
    height=600
)
```

```
alt.Chart(...)
```

It is not just bar plot that you can have fun with. Line plots are another interesting example.

For our simple line plot, we will need the poll data for a single state.

```
df_texas = df_polls[
    df_polls['state'] == 'Texas'
]

df_texas_bt = df_texas[
    (df_texas['candidate_name'] == 'Donald Trump') |
    (df_texas['candidate_name'] == 'Joseph R. Biden Jr.')
]

df_texas_bt.head()
```

	cycle	state	modeldate	candidate_name	pct_estimate	pct_trend_adjusted
7	2020	Texas	11/3/2020	Joseph R. Biden Jr.	47.46643	47.44781
63	2020	Texas	11/3/2020	Donald Trump	48.57118	48.58794
231	2020	Texas	11/2/2020	Joseph R. Biden Jr.	47.46643	47.44781
287	2020	Texas	11/2/2020	Donald Trump	48.57118	48.58794
455	2020	Texas	11/1/2020	Joseph R. Biden Jr.	47.45590	47.43400

The modeldate column is a string (object) and not date time. So we need to change that.

```
print('Before\n')
print(df_texas_bt.dtypes)
df_texas_bt['date'] = pd.to_datetime(df_texas_bt.loc[:, 'modeldate'], format='%m/%d/%Y')
print('\nAfter\n')
print(df_texas_bt.dtypes)
```

Before

```
cycle           int64
state            object
modeldate        object
candidate_name   object
pct_estimate     float64
pct_trend_adjusted float64
dtype: object
```

After

```
cycle           int64
state            object
modeldate        object
candidate_name   object
pct_estimate     float64
pct_trend_adjusted float64
date            datetime64[ns]
dtype: object
```

```
/var/folders/7v/z19mv52s3ls94kntlt_19ryh0000gq/T/ipykernel_4898/3000856052.py:3: SettingWithCopyWarning
  A value is trying to be set on a copy of a slice from a DataFrame.
  Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#inplace-operations
df_texas_bt['date'] = pd.to_datetime(df_texas_bt.loc[:, 'modeldate'], format='%m/%d/%Y')
```

Create our line plot.

```
alt.Chart(df_texas_bt).mark_line().encode(
    y=alt.Y('pct_estimate', scale=alt.Scale(domain=[42, 53])),
    x='date',
    color='candidate_name')
```

```
alt.Chart(...)
```

Sometimes multiple axis are used for each line, or in a combined line and bar plot.

The example [here](#) uses a dataframe with a column for each line. Our data does not have that.

```
df_texas_bt
our_df = df_texas_bt[['candidate_name', 'pct_estimate', 'date']]
our_df
```

	candidate_name	pct_estimate	date
7	Joseph R. Biden Jr.	47.46643	2020-11-03
63	Donald Trump	48.57118	2020-11-03
231	Joseph R. Biden Jr.	47.46643	2020-11-02
287	Donald Trump	48.57118	2020-11-02
455	Joseph R. Biden Jr.	47.45590	2020-11-01
...
28931	Donald Trump	49.09724	2020-02-29
28963	Joseph R. Biden Jr.	45.30901	2020-02-28
28995	Donald Trump	49.09676	2020-02-28
29027	Joseph R. Biden Jr.	45.30089	2020-02-27
29058	Donald Trump	49.07925	2020-02-27

Pivot table allows us to reshape our dataframe.

```
our_df = pd.pivot_table(our_df, index=['date'], columns = 'candidate_name')
our_df.columns = our_df.columns.to_series().str.join('_')
our_df.head()
```

date	pct_estimate_Donald Trump	pct_estimate_Joseph R. Biden Jr.
2020-02-27	49.07925	45.30089
2020-02-28	49.09676	45.30901
2020-02-29	49.09724	45.30896
2020-03-01	49.09724	45.30895
2020-03-02	48.91861	45.37694

Date here is the dataframe index. We want it to be a column.

```
our_df['date1'] = our_df.index
our_df.columns = ['Trump', 'Biden', 'date1']
our_df.head()
```

	Trump	Biden	date1
date			
2020-02-27	49.07925	45.30089	2020-02-27
2020-02-28	49.09676	45.30901	2020-02-28
2020-02-29	49.09724	45.30896	2020-02-29
2020-03-01	49.09724	45.30895	2020-03-01
2020-03-02	48.91861	45.37694	2020-03-02

Creating our new plot, to fool all those people who expect Trump to win in Texas.

```
base = alt.Chart(our_df).encode(
    alt.X('date1')
)

line_A = base.mark_line(color="#5276A7").encode(
    alt.Y('Trump', axis=alt.Axis(titleColor="#5276A7"), scale=alt.Scale(domain=[42,53]))
)

line_B = base.mark_line(color="#F18727").encode(
    alt.Y('Biden', axis=alt.Axis(titleColor="#F18727"), scale=alt.Scale(domain=[35,53]))
)

alt.layer(line_A, line_B).resolve_scale(y='independent')

alt.LayerChart(...)
```

Did you see what I did there?

Of course, mixed axis plots are rarely purely line plots. Instead they can be mixes of different axis. For these and other plotting mistakes, the economist has a nice article [here](#). You may want to try some of these plots with this data set or the world indicators dataset from a few weeks ago.

30 IM939 - Lab 6 - Part 3 - Choropleths

A visualisation often shown is a choropleth. This is a series of spatial polygons (such as states in the USA) which are coloured by a feature. Here we will look at creating choropleths of polling data in the recent USA election.

Load in two datasets. One contains the geospatial polygons of the states in America and the other is the polling data we used in the last notebook.

```
import geopandas as gpd
import pandas as pd
import altair as alt

geo_states = gpd.read_file('data/gz_2010_us_040_00_500k.json')
df_polls = pd.read_csv('data/presidential_poll_averages_2020.csv')
```

Filter our poll data to a specific date.

```
df_nov = df_polls[
    (df_polls.modeldate == '11/3/2020')
]

df_nov_states = df_nov[
    (df_nov.candidate_name == 'Donald Trump') |
    (df_nov.candidate_name == 'Joseph R. Biden Jr.')
]

df_nov_states
```

	cycle	state	modeldate	candidate_name	pct_estimate	pct_trend_adjusted
0	2020	Wyoming	11/3/2020	Joseph R. Biden Jr.	30.81486	30.82599
1	2020	Wisconsin	11/3/2020	Joseph R. Biden Jr.	52.12642	52.09584
2	2020	West Virginia	11/3/2020	Joseph R. Biden Jr.	33.49125	33.51517
3	2020	Washington	11/3/2020	Joseph R. Biden Jr.	59.34201	59.39408
4	2020	Virginia	11/3/2020	Joseph R. Biden Jr.	53.74120	53.72101
...

	cycle	state	modeldate	candidate_name	pct_estimate	pct_trend_adjusted
107	2020	California	11/3/2020	Donald Trump	32.28521	32.43615
108	2020	Arkansas	11/3/2020	Donald Trump	58.39097	58.94886
109	2020	Arizona	11/3/2020	Donald Trump	46.11074	46.10181
110	2020	Alaska	11/3/2020	Donald Trump	50.99835	51.23236
111	2020	Alabama	11/3/2020	Donald Trump	57.36153	57.36126

The geo_states variable has polygons for each state.

```
geo_states.head()
```

	GEO_ID	STATE	NAME	LSAD	CENSUSAREA	geometry
0	0400000US23	23	Maine		30842.923	MULTIPOLYGON (((-67.61976 44.51...
1	0400000US25	25	Massachusetts		7800.058	MULTIPOLYGON ((((-70.83204 41.60...
2	0400000US26	26	Michigan		56538.901	MULTIPOLYGON ((((-88.68443 48.11...
3	0400000US30	30	Montana		145545.801	POLYGON ((-104.05770 44.99743, -1...
4	0400000US32	32	Nevada		109781.180	POLYGON ((-114.05060 37.00040, -1...

```
alt.Chart(geo_states, title='US states').mark_geoshape().encode(
  ).properties(
    width=500,
    height=300
  ).project(
    type='albersUsa'
)
```

```
alt.Chart(...)
```

We want to put the percentage estimates for each candidate onto the map. First, let us create a dataframe containing the data for each candidate.

```
# Create separate data frame for trump and biden
trump_data = df_nov_states[
  df_nov_states.candidate_name == 'Donald Trump'
]

biden_data = df_nov_states[
  df_nov_states.candidate_name == 'Joseph R. Biden Jr.'
```

```
]
```

Our spatial and poll data have the name of the state in common, but their columns have different names. We could rename the column so it is the same in all cases and then merge (see commented code below)

```
# Uncomment below to see the effect. This produces an almost identical geodataframe to code above

# Rename column names.
# trump_data.columns = ['cycle', 'NAME', 'modeldate', 'candidate_name', 'pct_estimate', 'pct']
# biden_data.columns = ['cycle', 'NAME', 'modeldate', 'candidate_name', 'pct_estimate', 'pct']

# We can join the geospatial and poll data using the NAME column (the name of the state).
# geo_states_trump = geo_states.merge(trump_data, on = 'NAME')
# geo_states_biden = geo_states.merge(biden_data, left_on = 'NAME', right_on = 'state')
```

We can join the geospatial and poll data using different column names by using `left_on` for the left data (usually the geodataframe) and `right_on` for the right dataframe.

```
# Add the poll data
geo_states_trump = geo_states.merge(trump_data, left_on = 'NAME', right_on = 'state')
geo_states_biden = geo_states.merge(biden_data, left_on = 'NAME', right_on = 'state')

geo_states_trump.head()
```

	GEO_ID	STATE	NAME	LSAD	CENSUSAREA	geometry
0	0400000US23	23	Maine		30842.923	MULTIPOLYGON (((-67.61976 44.51...
1	0400000US25	25	Massachusetts		7800.058	MULTIPOLYGON (((-70.83204 41.60...
2	0400000US26	26	Michigan		56538.901	MULTIPOLYGON (((-88.68443 48.11...
3	0400000US30	30	Montana		145545.801	POLYGON ((-104.05770 44.99743, -1...
4	0400000US32	32	Nevada		109781.180	POLYGON ((-114.05060 37.00040, -1...

```
geo_states_biden.head()
```

	GEO_ID	STATE	NAME	LSAD	CENSUSAREA	geometry
0	0400000US23	23	Maine		30842.923	MULTIPOLYGON (((-67.61976 44.51...
1	0400000US25	25	Massachusetts		7800.058	MULTIPOLYGON (((-70.83204 41.60...
2	0400000US26	26	Michigan		56538.901	MULTIPOLYGON (((-88.68443 48.11...
3	0400000US30	30	Montana		145545.801	POLYGON ((-104.05770 44.99743, -1...

Joe Biden is clearly winning. Can we make it look like he is not?

We can plot this specifying the feature to use for our colour.

```
alt.Chart(geo_states_trump, title='Poll estimate for Donald Trump on 11/3/2020').mark_geos  
    .color='pct_estimate',  
    tooltip=['NAME', 'pct_estimate']  
).properties(  
    width=500,  
    height=300  
).project(  
    type='albersUsa'  
)  
  
t.Chart(...)
```

To smooth out any differences we can bin our data.

```
alt.Chart(geo_states_trump, title='Poll estimate for Donald Trump on 11/3/2020').mark_geos  
    .color('pct_estimate', bin=alt.Bin(step=35)),  
    tooltip=['NAME', 'pct_estimate']  
).properties(  
    width=500,  
    height=300  
).project(  
    type='albersUsa'  
)  
  
t.Chart(...)
```

How would you interpret the plot above?

What about if we increase the binstep so we have more bins?

```
alt.Chart(geo_states_trump, title='Poll estimate for Donald Trump on 11/3/2020').mark_geos  
    alt.Color('pct_estimate', bin=alt.Bin(step=5)),  
    tooltip=['NAME', 'pct_estimate']
```

```
  ).properties(
    width=500,
    height=300
  ).project(
    type='albersUsa'
)

alt.Chart(...)
```

Perhaps try different step sizes for the bins and consider how bins can shape our interpretation of the data. What would happen if plots with different bin sizes were placed side to side.

To add further confusion, what happens when we log scale the data?

```
alt.Chart(geo_states_trump, title='Poll estimate for Donald Trump on 11/3/2020').mark_geoshape
  alt.Color('pct_estimate', bin=alt.Bin(step=5), scale=alt.Scale(type='log')),
  tooltip=['NAME', 'pct_estimate']
).properties(
  width=500,
  height=300
).project(
  type='albersUsa'
)

alt.Chart(...)
```

vs

```
alt.Chart(geo_states_biden, title='Poll estimate for Joe Biden on 11/3/2020').mark_geoshape
  alt.Color('pct_estimate', bin=alt.Bin(step=5), scale=alt.Scale(type='log')),
  tooltip=['NAME', 'pct_estimate']
).properties(
  width=500,
  height=300
).project(
  type='albersUsa'
)

alt.Chart(...)
```

What is happening here?!?!

Next up, what about the colours we use and the range of values assigned to each color? Code inspired by/taken from [here](#).

```
alt.Chart(geo_states_trump, title='Poll estimate for Donald Trump on 11/3/2020').mark_geoshape()
    alt.Color('pct_estimate',
        scale=alt.Scale(type="linear",
            domain=[10, 40, 50, 55, 60, 61, 62],
            range=["#414487", "#414487",
                "#355f8d", "#355f8d",
                "#2a788e",
                "#fde725", "#fde725"])),
    tooltip=['NAME', 'pct_estimate']
).properties(
    width=500,
    height=300
).project(
    type='albersUsa'
)

alt.Chart(...)
```

Compare that with

```
alt.Chart(geo_states_trump, title='Poll estimate for Donald Trump on 11/3/2020').mark_geoshape()
    alt.Color('pct_estimate',
        scale=alt.Scale(type="linear",
            domain=[10, 20, 30, 35, 68, 70, 100],
            range=["#414487", "#414487",
                "#7ad151", "#7ad151",
                "#bddf26",
                "#fde725", "#fde725"])),
    tooltip=['NAME', 'pct_estimate']
).properties(
    width=500,
    height=300
).project(
    type='albersUsa'
)

alt.Chart(...)
```

My goodness! So what have we played around with?

- Transforming our scale using log
- Binning our data to smooth out variances
- Altering our colour scheme and the ranges for each colour

... what about if we remove the legend?

```
alt.Chart(geo_states_trump, title='Poll estimate for Donald Trump on 11/3/2020').mark_geos  
    alt.Color('pct_estimate',  
        scale=alt.Scale(type="linear",  
            domain=[10, 20, 30, 35, 68, 70, 100],  
            range=["#414487", "#414487",  
                "#7ad151", "#7ad151",  
                "#bddf26",  
                "#fde725", "#fde725"])),  
        legend=None),  
    tooltip=['NAME', 'pct_estimate']  
).properties(  
    width=500,  
    height=300  
).project(  
    type='albersUsa'  
)  
  
alt.Chart(...)
```

Good luck trying to interpret that. Though we often see maps without legends and with questionable colour schemes on TV.

How do you think choropleths should be displayed? What information does a use need to understand the message communicated in these plots?

31 IM939 - Lab 6 - Part 4 - Exercise

We have covered a lot of visualisations in the lab today.

For this exercise you should:

1. Identity a trend in your data. For instance, the higher poll scores for a presidential candidate in a particular state.
2. Create a visualisation which communicates that trend well. Consider why it is a good visualisation. Could someone else misunderstand the visualisation?
3. Create another visualisation. The goal of the second visualisation is to be intentionally misleading.

Please work on this in the same groups as the exercises in the previous weeks. There are sections below which are to guide you. Feel free to disregard these if you want to work another way.

You can use any dataset from this or previous weeks. Also, feel free to explore new datasets from FiveThirtyFive, the sklean datasets or the UCI machine learning datasets.

You may find the following helpful:

- [Altair examples](#). In general, the Altair documentation can be tricky to understand. I tend to follow the examples or google round for solutions.
- [Seaborn examples](#).

Load in your libraries and datasets

Subset or clean your data

Carry out an analysis if required. E.g., are you running a PCA or other dimension reduction, or a linear regression to plot?

Create a visualisation which clearly show the trend you would like to show in your data.

Why did you choose this visualisation? Do you think other will clearly see the trend you have identified?

Create a visualisation of this trend which you think will mislead the user?

How do you think this will mislead the user?

32 IM939 - Lab 6 Part 5 Simpson's Paradox

In the session 6 (week 7) video we discussed the Simpson's Paradox. You can explore some case studies in the **applet** by clicking [here](#)

Here we are going to look at the case study of “**Expenditure data for developmentally-disabled California residents**”. This dataset was adjusted (the original dataset you can find [here](#)) in order to explain the Simpson’s Paradox. You can read the research paper “Simpson’s Paradox A Data Set and Discrimination Case Study” that uses and explains this dataset [here](#). There is also a [documentation of the dataset](#)

32.1 1 Data

Here are the key information from the dataset documentation file (every time I use “ ” below it is a cite from the dataset file). ### Abstract: “The State of California Department of Developmental Services (DDS) is responsible for allocating funds that support over 250,000 developmentally-disabled residents (e.g., intellectual disability, cerebral palsy, autism, etc.), called here also consumers. The dataset represents a sample of 1,000 of these consumers. Biographical characteristics and expenditure data (i.e., the dollar amount the State spends on each consumer in supporting these individuals and their families) are included in the data set for each consumer.

32.1.1 Source:

The data set originates from DDS’s “Client Master File.” In order to remain in compliance with California State Legislation, the data have been altered to protect the rights and privacy of specific individual consumers. The data set is designed to represent a sample of 1,000 DDS consumers.

32.1.2 Variable Descriptions:

A header line contains the name of the variables. There are no missing values.

Id: 5-digit, unique identification code for each consumer (similar to a social security number)

Age Cohort: Binned age variable represented as six age cohorts (0-5, 6-12, 13-17, 18-21, 22-50, and 51+)

Age: Unbinned age variable

Gender: Male or Female

Expenditures: Dollar amount of annual expenditures spent on each consumer

Ethnicity: Eight ethnic groups (American Indian, Asian, Black, Hispanic, Multi-race, Native Hawaiian, Other, and White non-Hispanic).

32.1.3 Research problem

The data set and case study are based on a real-life scenario where there was a claim of discrimination based on ethnicity. The exercise highlights the importance of performing rigorous statistical analysis and how data interpretations can accurately inform or misguide decision makers.” (Taylor, Mickel 2014)

32.2 2 Reading the dataset

You should know the Pandas library already from the lab 1 with James. Here we are going to use it to explore the data and for pivot tables. In the folder you downloaded from the Moodle you have a dataset called ‘Lab 6 - Paradox Dataset’.

```
import pandas as pd
df = pd.read_excel('data/Paradox_Dataset.xlsx')
```

A reminder: anything with a pd. prefix comes from pandas. This is particularly useful for preventing a module from overwriting inbuilt Python functionality.

Let’s have a look at our dataset

```
df
```

	Id	AgeCohort	Age	Gender	Expenditures	Ethnicity
0	10210	13-17	17	Female	2113	White not Hispanic
1	10409	22-50	37	Male	41924	White not Hispanic
2	10486	0-5	3	Male	1454	Hispanic
3	10538	18-21	19	Female	6400	Hispanic
4	10568	13-17	13	Male	4412	White not Hispanic
...

	Id	AgeCohort	Age	Gender	Expenditures	Ethnicity
995	99622	51 +	86	Female	57055	White not Hispanic
996	99715	18-21	20	Male	7494	Hispanic
997	99718	13-17	17	Female	3673	Multi Race
998	99791	6-12	10	Male	3638	Hispanic
999	99898	22-50	23	Male	26702	White not Hispanic

We have 6 columns (variables) in 1000 rows. Let's see what type of object is our dataset and what types of objects are in the dataset.

```
type(df)
```

```
pandas.core.frame.DataFrame
```

32.3 3 Exploring data

32.3.1 Missing values

Let's check if we have any missing data

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
 ---  --          --          --      
 0   Id           1000 non-null   int64  
 1   AgeCohort    1000 non-null   object  
 2   Age          1000 non-null   int64  
 3   Gender        1000 non-null   object  
 4   Expenditures 1000 non-null   int64  
 5   Ethnicity     1000 non-null   object  
dtypes: int64(3), object(3)
memory usage: 47.0+ KB
```

The above table shows that we have 1000 observations for each of 6 columns.

Let's see if there are any unexpected values.

```
import numpy as np
np.unique(df.AgeCohort)

array([' 0-5', ' 51 +', '13-17', '18-21', '22-50', '6-12'], dtype=object)

np.unique(df.Age)

array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 44, 45, 46, 48, 51, 52, 53, 54,
       55, 56, 57, 59, 60, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
       74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 85, 86, 88, 89, 90, 91, 94,
       95])

np.unique(df.Gender)

array(['Female', 'Male'], dtype=object)

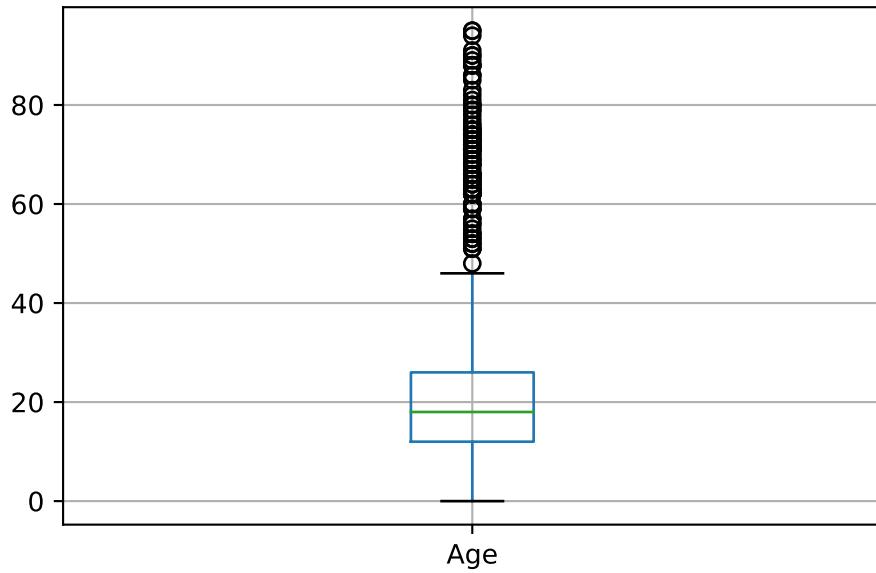
np.unique(df.Ethnicity)

array(['American Indian', 'Asian', 'Black', 'Hispanic', 'Multi Race',
       'Native Hawaiian', 'Other', 'White not Hispanic'], dtype=object)
```

There aren't any unexpected values in either of these 4 variables. We didn't run this command for Expenditures on purpose, as this would return us too many values. An easier way to check this variable would be just a boxplot.

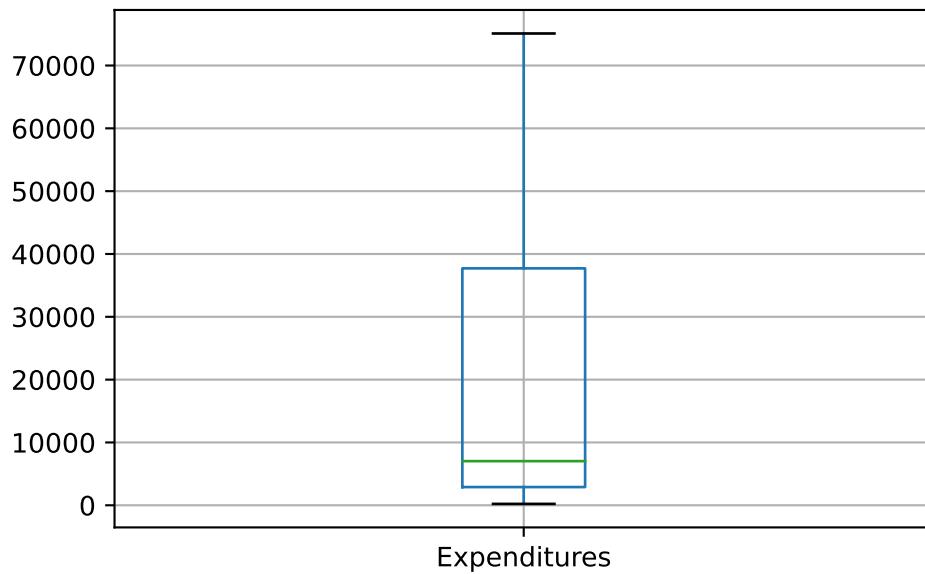
```
df.boxplot(column=['Age'])
```

```
<Axes: >
```



```
df.boxplot(column=['Expenditures'])
```

```
<Axes: >
```



Let's see a summary of data types we have here.

32.3.2 Data types

```
df.dtypes
```

```
Id           int64
AgeCohort    object
Age           int64
Gender         object
Expenditures  int64
Ethnicity     object
dtype: object
```

We are creating a new categorical column `cat_AgeCohort` that would make our work a bit easier later. You can read more [here](#)

```
df['cat_AgeCohort'] = pd.Categorical(df['AgeCohort'],
                                      ordered=True,
                                      categories=['0-5', '6-12', '13-17', '18-21', '22-50',
```

Here `int64` mean ‘a 64-bit integer’ and ‘`object`’ are strings. This gives you also a hint they are different types of variables. The ‘bit’ refers to how long and precise the number is. Pandas uses data types from numpy ([pandas documentation](#), [numpy documentation](#)). In our dataset three variables are numeric: `Id`, `age` are ordinal variables, `Expenditures` is a scale variable. `AgeCohort` is categorical and `Gender` and `Ethnicity` are nominal.

For that reason ‘`data.describe`’ will bring us a summary of numeric variables only.

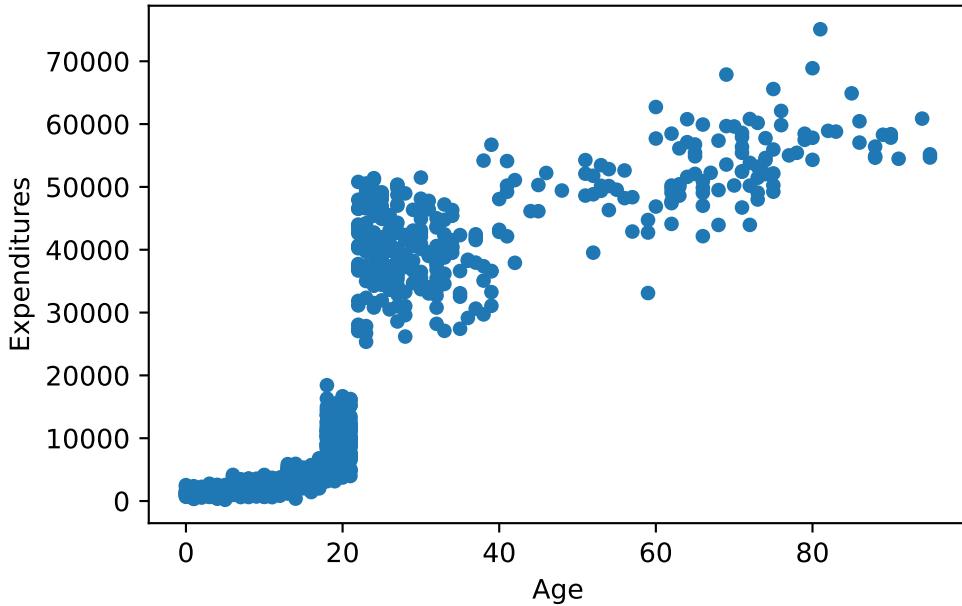
```
df.describe()
```

	Id	Age	Expenditures
count	1000.000000	1000.000000	1000.000000
mean	54662.846000	22.800000	18065.786000
std	25643.673401	18.462038	19542.830884
min	10210.000000	0.000000	222.000000
25%	31808.750000	12.000000	2898.750000
50%	55384.500000	18.000000	7026.000000
75%	76134.750000	26.000000	37712.750000
max	99898.000000	95.000000	75098.000000

It doesn’t make sense to plot not numeric variables or ids. That’s why we are going to just plot `age` and `Expenditures`.

```
df.plot(x = 'Age', y = 'Expenditures', kind='scatter')
```

```
<Axes: xlabel='Age', ylabel='Expenditures'>
```



The pattern of data is very interesting, especially around x-values of ca. 25. The research paper can bring us more clarification.

32.3.3 Age

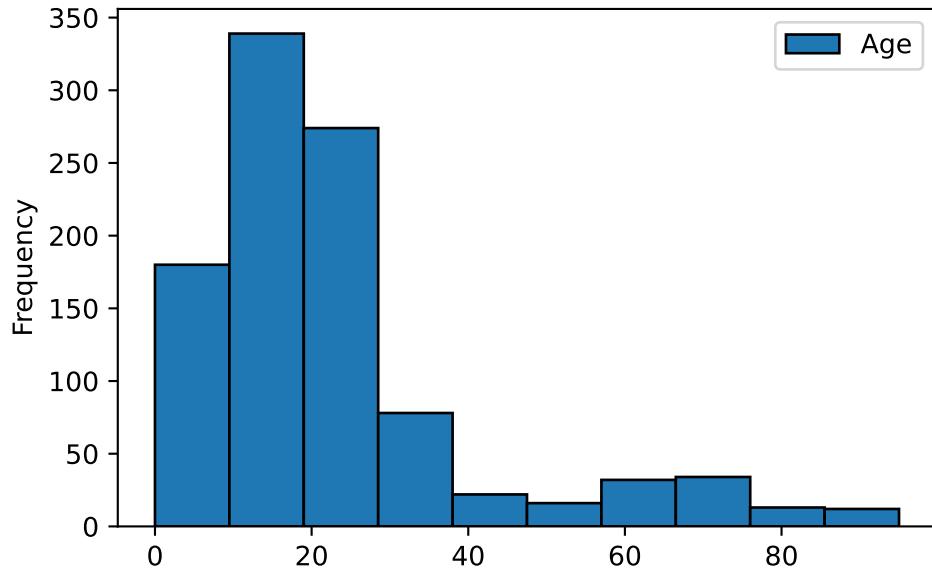
The crucial factor in this case study is age: “As consumers get older, their financial needs increase as they move out of their parent’s home, etc. Therefore, it is expected that expenditures for older consumers will be higher than for the younger consumers”

In the dataset we have two age variables that both refer to the same information - age of consumers. They are saved as two distinct data types: binned ‘AgeCohort’ and unbinned ‘Age’.

Age categories If you look at the binned one you will notice that the categories are somewhat interesting:

```
df[['Age']].plot(kind='hist', ec='black')
```

```
<Axes: ylabel='Frequency'>
```



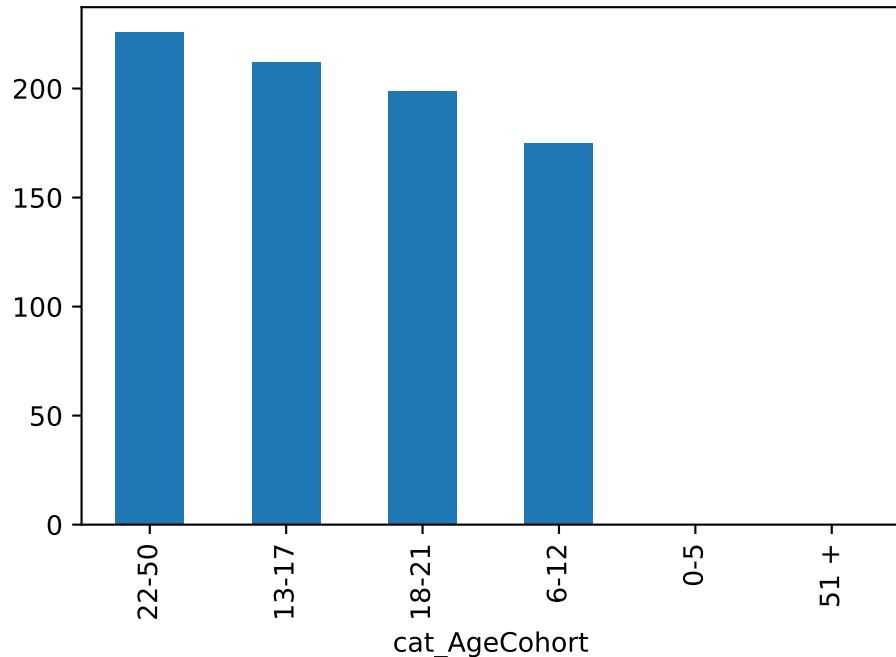
```
df["AgeCohort"].describe() #we will receive the output for the categorical variable "AgeCohort"
df['cat_AgeCohort'].describe()
```

```
count      812
unique       4
top      22-50
freq      226
Name: cat_AgeCohort, dtype: object
```

Here we will run a bar plot of age categories.

```
df['cat_AgeCohort'].value_counts().plot(kind="bar")
```

```
<Axes: xlabel='cat_AgeCohort'>
```



The default order of plot elements is ‘value count’. For the age variable it might be more useful to look at the order chronologically.

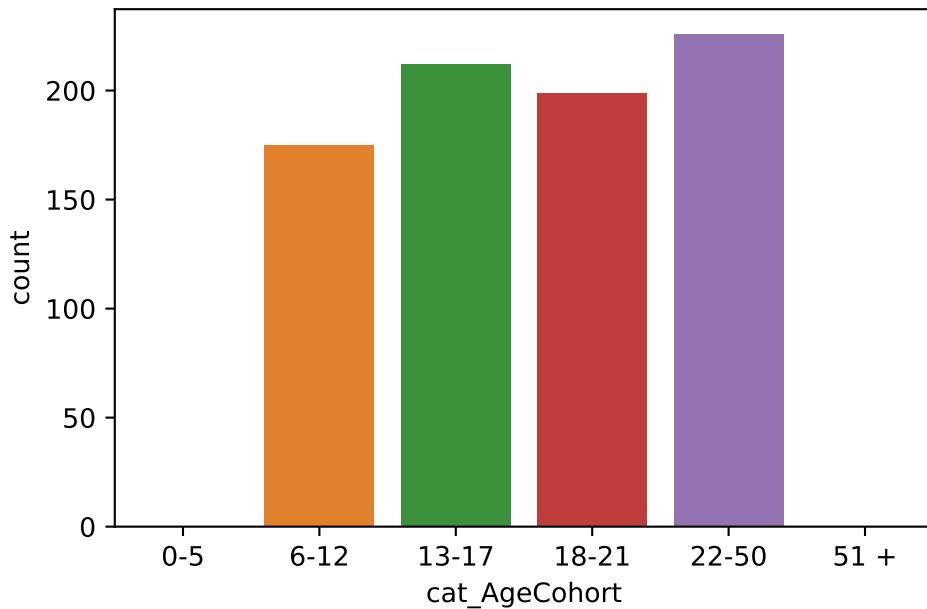
```
#using sns.countplot from seaborn we will plot AgeCohort
#the order in plotting this variable is really crucial, we want to have sorted by age cate
import seaborn as sns
import matplotlib.pyplot as plt

#here is without sorting / ordering
#sns.countplot(x="AgeCohort", data=df)

#here we plot the variable with sorting
sns.countplot(x="cat_AgeCohort", data=df)

#You can try playing with the commands below too:
#sns.countplot(x="AgeCohort", data=df, order=df['AgeCohort'].value_counts().index)
#sns.countplot(x="AgeCohort", data=df, order=['0-5', '6-12', '13-17', '18-21', '22-50', '51+'])

<Axes: xlabel='cat_AgeCohort', ylabel='count'>
```



Why would the data be binned in such “uneven” categories like ‘0-5 years’, ‘6-12’ and ‘22-50’? Instead of even categories e.g. ‘0-10’, ‘11-20’, ‘21-30’ etc. or every 5 years ‘0-5’, ‘6-10’ etc.?

Here the age cohorts **were allocated based on the theory**, rather than based on data (this way we would have even number of people in each category) or based on logical age categories, e.g. every 5 or 10 years.

The authors explain: “The cohorts are established based on the amount of financial support typically required during a particular life phase (...) The 0-5 cohort (preschool age) has the fewest needs and requires the least amount of funding (...) Those in the 51+ cohort have the most needs and require the most amount of funding”. You can read in more details in the paper.

32.4 4 Exploratory analysis

The research question is: *are any demographics discriminated in distributions of the funds?*

Following the authors: “Discrimination exists in this case study if the amount of expenditures for a typical person in a group of consumers that share a common attribute (e.g., gender, ethnicity, etc.) is significantly different when compared to a typical person in another group. For example, discrimination based on gender would occur if the expenditures for a typical female are less than the amount for a typical male.”

We are going to examine the data using plots for categorical data and pivot tables (cross-tables) with means. “Pivot table reports are particularly useful in narrowing down larger data sets or analyzing relationships between data points.” Pivot tables will help you understand what is Simpson’s Paradox.

32.4.1 Age x expenditures

Let’s see how expenditures are distributed across age groups.

We are going to use a *swarm plot* which I believe works well here to notice the paradox and “the points are adjusted (only along the categorical axis) so that they don’t overlap. This gives a better representation of the distribution of values, but it does not scale well to large numbers of observations. A swarm plot can be drawn on its own, but it is also a good complement to a box or violin plot in cases where you want to show all observations along with some representation of the underlying distribution.” Read more [here](#)

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.catplot(x="AgeCohort", y="Expenditures", kind="swarm", data=df)
#you can also do a boxplot if you change kind="box"

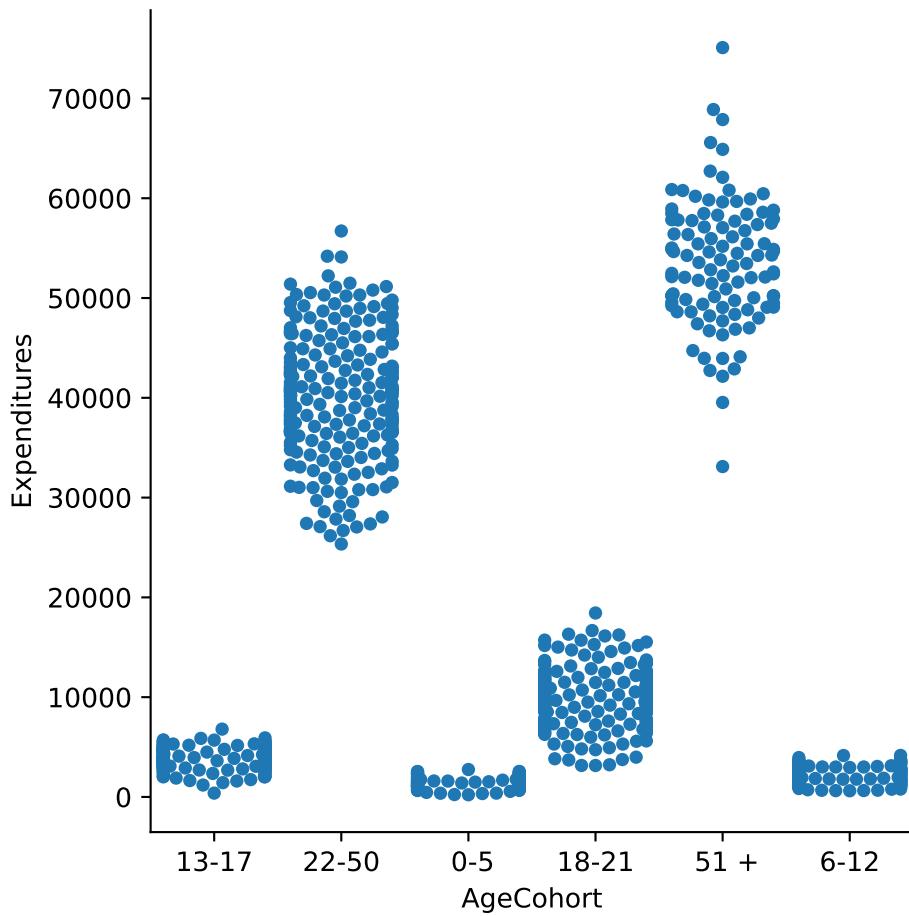
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354
    warnings.warn(msg, UserWarning)

/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354
    warnings.warn(msg, UserWarning)

/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354
    warnings.warn(msg, UserWarning)
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354
    warnings.warn(msg, UserWarning)
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354
```

```
warnings.warn(msg, UserWarning)
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354:
  warnings.warn(msg, UserWarning)
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354:
  warnings.warn(msg, UserWarning)

/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354:
  warnings.warn(msg, UserWarning)
```



32.4.2 Ethnicity

Ethnicity could be another discriminating factor. Let's check this here too by plotting expenditures by ethnicity.

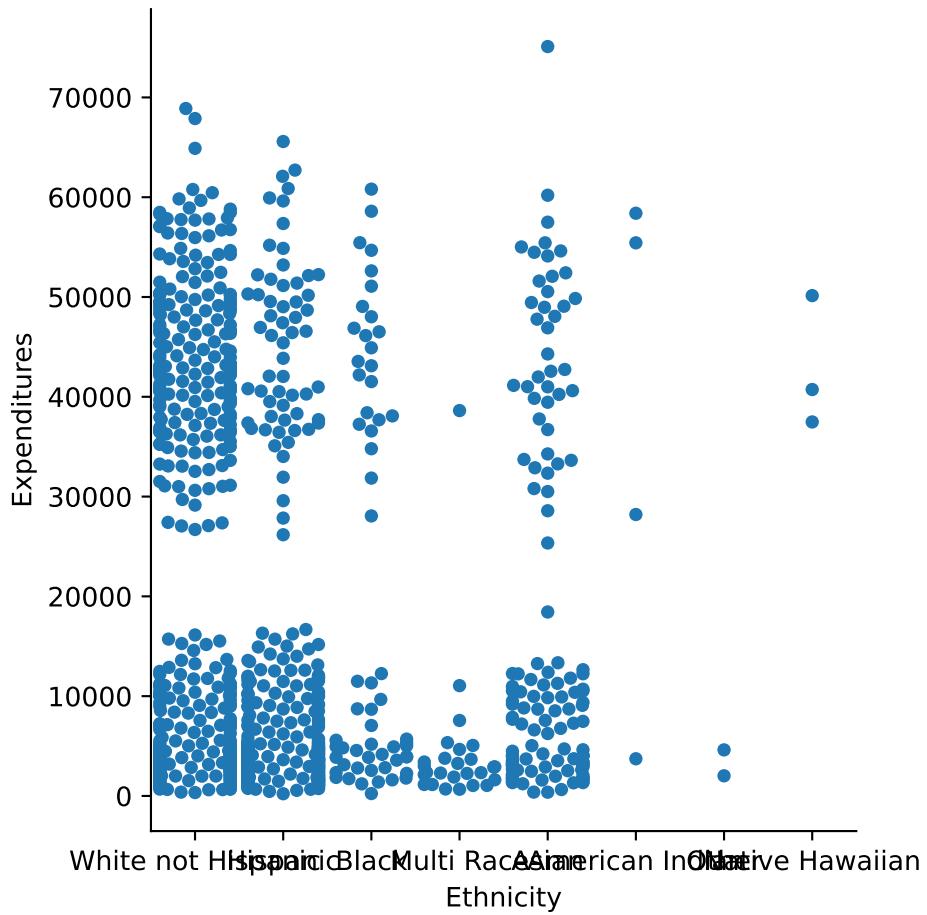
These groups reflect the demographic profile of the State of California.

```
sns.catplot(x="Ethnicity", y="Expenditures", kind="swarm", data=df)

/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354:
  warnings.warn(msg, UserWarning)

/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354:
  warnings.warn(msg, UserWarning)
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354:
  warnings.warn(msg, UserWarning)
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354:
  warnings.warn(msg, UserWarning)
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354:
  warnings.warn(msg, UserWarning)

/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354:
  warnings.warn(msg, UserWarning)
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354:
  warnings.warn(msg, UserWarning)
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354:
  warnings.warn(msg, UserWarning)
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354:
  warnings.warn(msg, UserWarning)
```



32.4.3 Gender

Gender could have been another discriminating factor (as gender based discrimination is also very common). It is not the case here. See below plots to confirm these. We are plotting expenditures by gender.

```

import seaborn as sns
import matplotlib.pyplot as plt

#sns.catplot(x="Gender", y="Expenditures", kind="swarm", data=df)
#you can create even a nicer plot than for ethnicity, using tips here https://seaborn.pydata.org
#It's a combination of swarmplot and violin plot to show each observation along with a sum

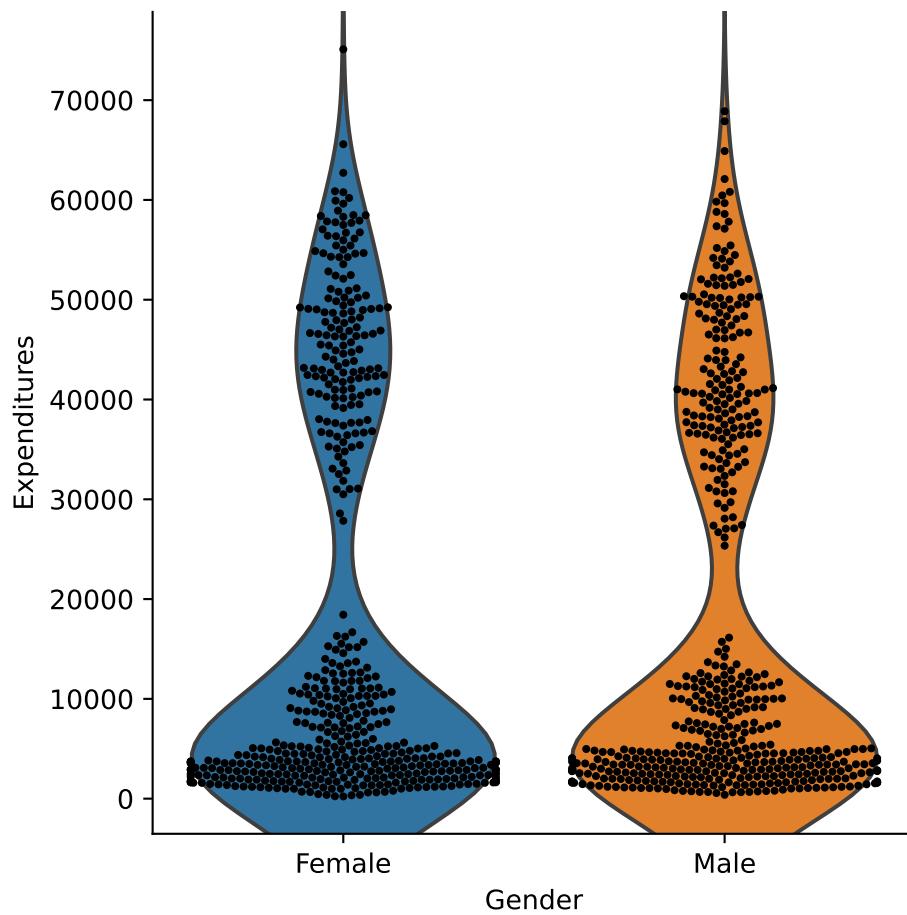
g = sns.catplot(x="Gender", y="Expenditures", kind="violin", inner=None, data=df)

```

```
sns.swarmplot(x="Gender", y="Expenditures", color="k", size=3, data=df, ax=g.ax)
```

```
<Axes: xlabel='Gender', ylabel='Expenditures'>
```

```
/Users/u2071219/anaconda3/envs/IM939/lib/python3.11/site-packages/seaborn/categorical.py:354:  
    warnings.warn(msg, UserWarning)
```



32.4.4 Mean Expenditures

This was a quick visual analysis. Let's check means to see how it looks like by age, ethnicity and gender. Why would it be also good to check medians here?

```

import pandas as pd
import numpy as np

#By default the aggregate function is mean

np.round(df.pivot_table(index=['Ethnicity'], values=['Expenditures']), 2)

```

Ethnicity	Expenditures
American Indian	36438.25
Asian	18392.37
Black	20884.59
Hispanic	11065.57
Multi Race	4456.73
Native Hawaiian	42782.33
Other	3316.50
White not Hispanic	24697.55

```
np.round(df.pivot_table(index=['Gender'], values=['Expenditures']), 2)
```

Gender	Expenditures
Female	18129.61
Male	18001.20

```
np.round(df.pivot_table(index=['cat_AgeCohort'], values=['Expenditures']), 2)
```

cat_AgeCohort	Expenditures
6-12	2226.86
13-17	3922.61
18-21	9888.54
22-50	40209.28

What do these tables tell us? There is much discrepancy in average results for ethnicity and age cohort. If we look at gender - there aren't many differences.

Please remember that in this case study “the needs for consumers increase as they become older which results in higher expenditures”. This would explain age discrepancies a bit, but what about ethnicity?

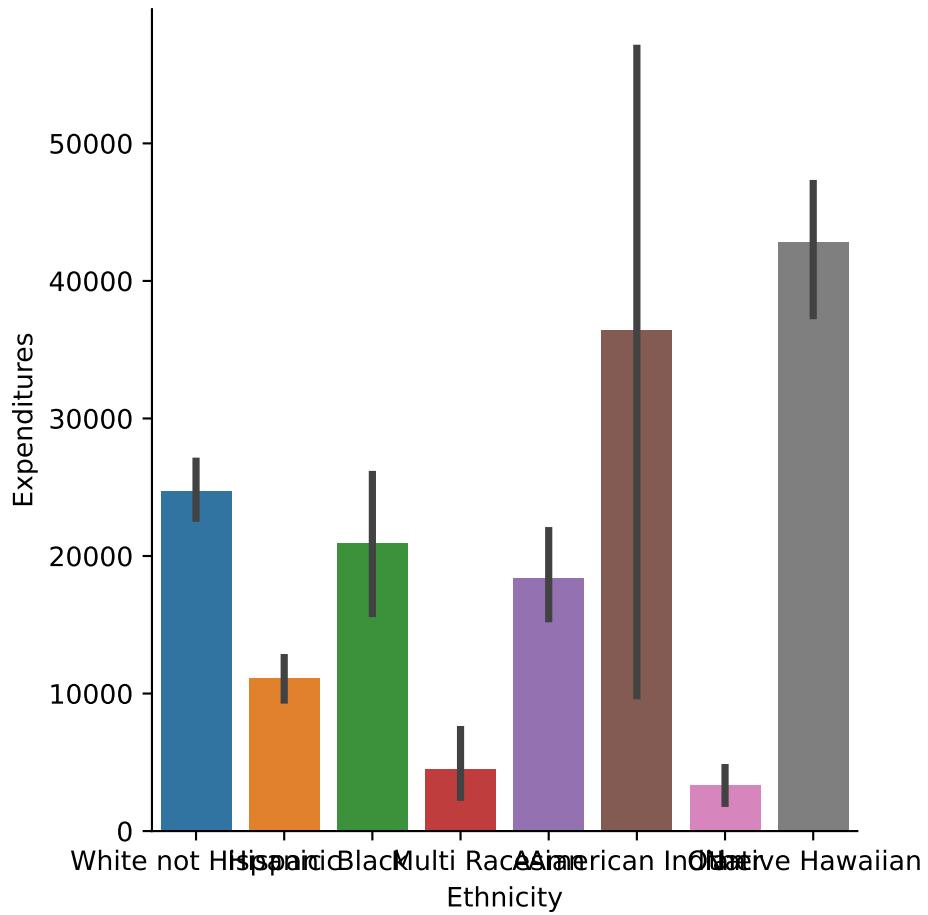
33 5 In-depth Analysis - Outliers

Let's try to go a bit more in-depth . We know that gender doesn't show many differences. In age there are not big differences except for one case. Let's focus on ethnicity then.

We are going to use Seaborn's 'catplot'. In the [documentation](#) we can read what are the error bars here: "In seaborn, the barplot() function operates on a full dataset and applies a function to obtain the estimate (taking the mean by default). When there are multiple observations in each category, it also uses bootstrapping to compute a confidence interval around the estimate, which is plotted using error bars."

```
sns.catplot(x="Ethnicity", y='Expenditures',  
            kind="bar", data=df)
```

```
#you can also run a nested table, but the chart might be more straightforward in analysis.  
#np.round(df.pivot_table(index=['cat_AgeCohort','Ethnicity'], values=['Expenditures']), 2)
```



```
np.round(df.pivot_table(index=['Ethnicity'], values=['Expenditures']), 2)
```

Ethnicity	Expenditures	
American Indian	36438.25	
Asian	18392.37	
Black	20884.59	
Hispanic	11065.57	
Multi Race	4456.73	
Native Hawaiian	42782.33	
Other	3316.50	
White not Hispanic	24697.55	

So there are big differences in the averages between ethnicities. Does it mean there is discrimination?

```
df.groupby('Ethnicity').count()
```

Ethnicity	Id	AgeCohort	Age	Gender	Expenditures	cat_AgeCohort
American Indian	4	4	4	4	4	2
Asian	129	129	129	129	129	108
Black	59	59	59	59	59	49
Hispanic	376	376	376	376	376	315
Multi Race	26	26	26	26	26	19
Native Hawaiian	3	3	3	3	3	2
Other	2	2	2	2	2	2
White not Hispanic	401	401	401	401	401	315

As you can see there are big sample size differences between ethnic groups.

What conclusions does it bring? There are 3 major ethnicities within the dataset: White non-Hispanic (40%), Hispanic (38%), Asian (13%). The sample sizes of other ethnicities are very small.

Please also remember that 1). We know it is representative data of the population of residents. So based on this data we can use inferential statistics (look up Week 03 slides if you need a reminder) and estimate results for the whole population of beneficiaries of California DDS.

2). Also, if you look into actual demographics of California State [here](#)

You will notice that the proportions of the state are similar to proportions of this case study. Hispanic and White non-Hispanic constitute a majority of California's population.

Let's focus on the top 2 biggest groups. We can see there is a difference in the average expenditures between the White non-Hispanic and Hispanic groups.

```
##selecting cases that are either 'Hispanic' or 'White non Hispanic'
Hispanic = df[(df["Ethnicity"] == 'Hispanic') | (df["Ethnicity"] == 'White not Hispanic')]
```

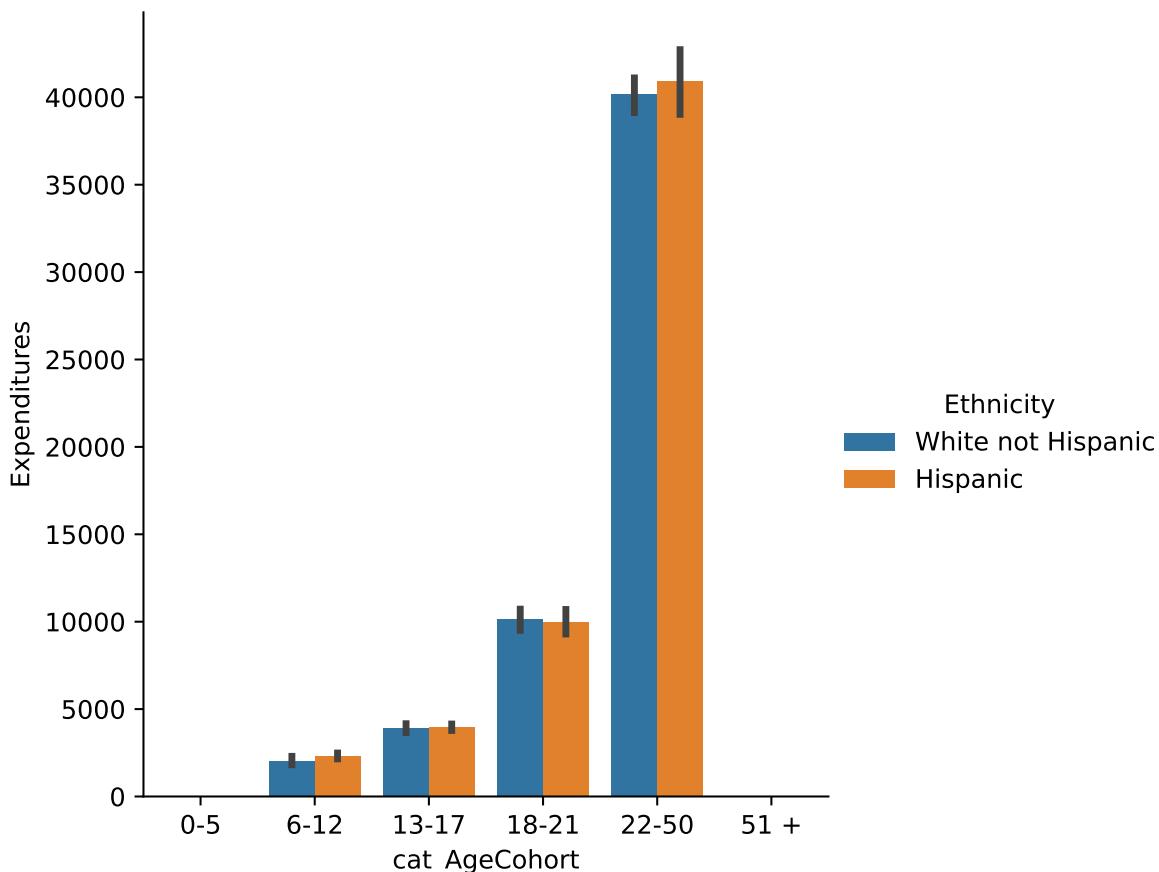
	Id	AgeCohort	Age	Gender	Expenditures	Ethnicity	cat_AgeCohort
0	10210	13-17	17	Female	2113	White not Hispanic	13-17
1	10409	22-50	37	Male	41924	White not Hispanic	22-50
2	10486	0-5	3	Male	1454	Hispanic	NaN

	Id	AgeCohort	Age	Gender	Expenditures	Ethnicity	cat_AgeCohort
3	10538	18-21	19	Female	6400	Hispanic	18-21
4	10568	13-17	13	Male	4412	White not Hispanic	13-17
...
992	99114	18-21	18	Male	5298	Hispanic	18-21
995	99622	51 +	86	Female	57055	White not Hispanic	NaN
996	99715	18-21	20	Male	7494	Hispanic	18-21
998	99791	6-12	10	Male	3638	Hispanic	6-12
999	99898	22-50	23	Male	26702	White not Hispanic	22-50

```
np.round(Hispanic.pivot_table(index=['Ethnicity', 'cat_AgeCohort'], values=['Expenditures']))
```

Ethnicity	cat_AgeCohort	Expenditures
		6-12
Hispanic	13-17	3955.28
	18-21	9959.85
	22-50	40924.12
	6-12	2052.26
White not Hispanic	13-17	3904.36
	18-21	10133.06
	22-50	40187.62

```
sns.catplot(x="cat_AgeCohort", y='Expenditures', hue="Ethnicity", kind="bar", data=Hispanic)
```



Let's get back to our original question : does discrimination exist in this case?

"Is the typical Hispanic consumer receiving fewer funds (i.e., expenditures) than the typical White non-Hispanic consumer? If a Hispanic consumer was to file for discrimination based upon ethnicity, s/he would more than likely be asked his/her age. Since the typical amount of expenditures for Hispanics (in all but one age cohort) is higher than the typical amount of expenditures for White non-Hispanics in the respective age cohort, the discrimination claim would be refuted".

This case study shows **Simpson's Paradox**. You may ask: "Why is the overall average for all consumers significantly different indicating ethnic discrimination of Hispanics, yet in all but one age cohort (18-21) the average of expenditures for Hispanic consumers are greater than those of the White non-Hispanic population?" Look at the table below.

```
pd.crosstab([Hispanic.cat_AgeCohort],Hispanic.Ethnicity)
```

Ethnicity cat_AgeCohort	Hispanic	White not Hispanic
6-12	91	46
13-17	103	67
18-21	78	69
22-50	43	133

Results

“There are more Hispanics in the youngest four age cohorts, while the White non-Hispanics have more consumers in the oldest two age cohorts. The two populations are close in overall counts (376 vs. 401). On top of this, consumers expenditures increase as they age to see the paradox.

Expenditure average for Hispanic consumers are higher in all but one of the age cohorts, but the trend reverses when the groups are combined resulting in a lower expenditure average for all Hispanic consumers when compared to all White non-Hispanics.”

“The overall Hispanic consumer population is a relatively younger when compared to the White non-Hispanic consumer population. Since the expenditures for younger consumers is lower, the overall average of expenditures for Hispanics (vs White non-Hispanics) is less.”

```
pd.crosstab(Hispanic.cat_AgeCohort,Hispanic.Ethnicity,
            normalize='columns')

# values=Hispanic.Ethnicity,aggfunc=sum,
```

Ethnicity cat_AgeCohort	Hispanic	White not Hispanic
6-12	0.288889	0.146032
13-17	0.326984	0.212698
18-21	0.247619	0.219048
22-50	0.136508	0.422222

34 6 Conclusions

34.1 Explanation

“This exercise is based on a real-life case in California. The situation involved an alleged case of discrimination privileging White non-Hispanics over Hispanics in the allocation of funds to over 250,000 developmentally-disabled California residents.

A number of years ago, an allegation of discrimination was made and supported by a univariate analysis that examined average annual expenditures on consumers by ethnicity. The analysis revealed that the average annual expenditures on Hispanic consumers was approximately one-third () of the average expenditures on White non-Hispanic consumers. (...) A bivariate analysis examining ethnicity and age (divided into six age cohorts) revealed that ethnic discrimination did not exist. Moreover, in all but one of the age cohorts, the trend reversed where the average annual expenditures on White non-Hispanic consumers were less than the expenditures on Hispanic consumers.”(Taylor, Mickel 2014)

When running the simple table with aggregated data, the discrimination in this case appeared evident. After running a few more detailed tables, it appears to be no evidence of discrimination based on this sample and the variables collected.

34.2 Takeaways

The example above concerns a crucial topic of discrimination. As you can see, data and statistics alone won't give us the answer. First results might give us a confusing result. Critical thinking is essential when working with data, in order to account for reasons not evident at the first sight. The authors remind us the following: 1) “outcome of important decisions (such as discrimination claims) are often heavily influenced by statistics and how an incomplete analysis may lead to poor decision making” 2) “importance of identifying and analyzing all sources of specific variation (i.e., potential influential factors) in statistical analyses”. This is something we already discussed in previous weeks, but it is never enough to stress it out”

34.2.1 *Additional Links

Some links regarding categorical data in Python for those interested:

https://pandas.pydata.org/pandas-docs/stable/user_guide/categorical.html#description

<https://pandas.pydata.org/pandas-docs/version/0.23.1/generated/pandas.DataFrame.plot.bar.html>

<https://seaborn.pydata.org/tutorial/categorical.html>

<https://seaborn.pydata.org/generated/seaborn.countplot.html>

Part VIII

Data Science & Society

35 IM939 - Lab 7 Part 1

In the session 7 (week 8) we discussed data and society: academic and practices discourse on the social, political and ethical aspects of data science, and discussed how one can responsibly carry out data science research on social phenomena, what ethical and social frameworks can help us to critically approach data science practices and its effects on society, and what are ethical practices for data scientists.

35.0.1 Datasets

Hate crimes a csv file <https://fivethirtyeight.com/features/higher-rates-of-hate-crimes-are-tied-to-income-inequality/>

OECD Poverty gap a csv file <https://data.oecd.org/inequality/poverty-rate.htm>

Poverty & Equity Data Portal <https://data.oecd.org/inequality/income-inequality.htm#indicator-chart>

<https://povertydata.worldbank.org/poverty/home/>

35.0.2 Further datasets

NHS multiple files The NHS inequality challenge <https://www.nuffieldtrust.org.uk/project/nhs-visual-data-challenge>

ONS

1. Gender Pay Gap <https://www.ons.gov.uk/employmentandlabourmarket/peopleinwork/earningsandworkingtime/guidanceandmethodology/genderpaygap/>
2. Health state life expectancies by Index of Multiple Deprivation (IMD 2015 and IMD 2019): England, all ages multiple publications <https://www.ons.gov.uk/peoplepopulationandcommunity/healthandlifeexpectancies/datasets/healthstatelifeexpectanciesbyindexofmultipledeprivationimdenglandallages>

35.0.3 Additional Readings

Indicators - critical reviews The Poverty of Statistics and the Statistics of Poverty
<https://www.tandfonline.com/doi/full/10.1080/01436590903321844?src=recsys>

Indicators in global health arguments: indicators are usually comprehensible to a small group of experts. Why use indicators then? „Because indicators used in global HIV finance offer openings for engagement to promote accountability (...) some indicators and data truly are better than others, and as they were all created by humans, they all can be deconstructed and remade in other forms” Davis, S. (2020). The Uncounted: Politics of Data in Global Health, Cambridge. doi:10.1017/9781108649544

Indicators - conceptualization

35.1 1 Hate Crimes

35.1.1 Source:

<https://github.com/fivethirtyeight/data/tree/master/hate-crimes>

35.1.2 Variables:

Header	Definition
state	State name
median_household_income	Median household income, 2016
share_unemployed_seasonal	Share of the population that is unemployed (seasonally adjusted), Sept. 2016
share_population_in_metro_areas	Share of the population that lives in metropolitan areas, 2015
share_population_with_high_school_degree	Share of adults 25 and older with a high-school degree, 2009
share_non_citizen	Share of the population that are not U.S. citizens, 2015
share_white_poverty	Share of white residents who are living in poverty, 2015
gini_index	Gini Index, 2015
share_non_white	Share of the population that is not white, 2015
share_voters_voted_trump	Share of 2016 U.S. presidential voters who voted for Donald Trump

Header	Definition
hate_crimes_per_100k_splc	Hate crimes per 100,000 population, Southern Poverty Law Center, Nov. 9-18, 2016
avg_hatecrimes_per_100k_fbi	Average annual hate crimes per 100,000 population, FBI, 2010-2015

35.2 2 Reading the dataset

```
import pandas as pd
df = pd.read_excel('data/hate_Crimes_v2.xlsx')
```

A reminder: anything with a pd. prefix comes from pandas. This is particularly useful for preventing a module from overwriting inbuilt Python functionality.

Let's have a look at our dataset

```
df.tail()
```

	NAME	median_household_income	share_unemployed_seasonal	share_population_in_metro
46	Virginia	66155	0.043	0.89
47	Washington	59068	0.052	0.86
48	West Virginia	39552	0.073	0.55
49	Wisconsin	58080	0.043	0.69
50	Wyoming	55690	0.040	0.31

```
type(df)
```

```
pandas.core.frame.DataFrame
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51 entries, 0 to 50
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  

```

```

----- ----- -----
0 NAME 51 non-null object
1 median_household_income 51 non-null int64
2 share_unemployed_seasonal 51 non-null float64
3 share_population_in_metro_areas 51 non-null float64
4 share_population_with_high_school_degree 51 non-null float64
5 share_non_citizen 48 non-null float64
6 share_white_poverty 51 non-null float64
7 gini_index 51 non-null float64
8 share_non_white 51 non-null float64
9 share_voters_voted_trump 51 non-null float64
10 hate_crimes_per_100k_splc 51 non-null float64
11 avg_hatecrimes_per_100k_fbi 51 non-null float64
dtypes: float64(10), int64(1), object(1)
memory usage: 4.9+ KB

```

35.3 3 Exploring data

35.3.1 Missing values

Let's explore the dataset

```
df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51 entries, 0 to 50
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
--- 
0   NAME            51 non-null      object 
1   median_household_income 51 non-null      int64  
2   share_unemployed_seasonal 51 non-null      float64
3   share_population_in_metro_areas 51 non-null      float64
4   share_population_with_high_school_degree 51 non-null      float64
5   share_non_citizen 48 non-null      float64
6   share_white_poverty 51 non-null      float64
7   gini_index       51 non-null      float64
8   share_non_white 51 non-null      float64
9   share_voters_voted_trump 51 non-null      float64
10  hate_crimes_per_100k_splc 51 non-null      float64
11  avg_hatecrimes_per_100k_fbi 51 non-null      float64

```

```
dtypes: float64(10), int64(1), object(1)
memory usage: 4.9+ KB
```

The above tables shows that we have some missing data for some of states. See below too.

```
df.isna().sum()
```

```
NAME          0
median_household_income 0
share_unemployed_seasonal 0
share_population_in_metro_areas 0
share_population_with_high_school_degree 0
share_non_citizen 3
share_white_poverty 0
gini_index 0
share_non_white 0
share_voters_voted_trump 0
hate_crimes_per_100k_splc 0
avg_hatecrimes_per_100k_fbi 0
dtype: int64
```

```
import numpy as np
np.unique(df.NAME)
```

```
array(['Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California',
       'Colorado', 'Connecticut', 'Delaware', 'District of Columbia',
       'Florida', 'Georgia', 'Hawaii', 'Idaho', 'Illinois', 'Indiana',
       'Iowa', 'Kansas', 'Kentucky', 'Louisiana', 'Maine', 'Maryland',
       'Massachusetts', 'Michigan', 'Minnesota', 'Mississippi',
       'Missouri', 'Montana', 'Nebraska', 'Nevada', 'New Hampshire',
       'New Jersey', 'New Mexico', 'New York', 'North Carolina',
       'North Dakota', 'Ohio', 'Oklahoma', 'Oregon', 'Pennsylvania',
       'Rhode Island', 'South Carolina', 'South Dakota', 'Tennessee',
       'Texas', 'Utah', 'Vermont', 'Virginia', 'Washington',
       'West Virginia', 'Wisconsin', 'Wyoming'], dtype=object)
```

There aren't any unexpected values in 'state'.

35.4 Mapping hate crime across the USA

```
#using James' code from the last lab: we need the geospatial polygons of the states in Am
import geopandas as gpd
import pandas as pd
import altair as alt

geo_states = gpd.read_file('data/gz_2010_us_040_00_500k.json')
#df = pd.read_excel('data/hate_Crimes_v2.xlsx')
geo_states.head()
```

	GEO_ID	STATE	NAME	LSAD	CENSUSAREA	geometry
0	0400000US23	23	Maine	30842.923	MULTIPOLYGON (((-67.61976 44.51	
1	0400000US25	25	Massachusetts	7800.058	MULTIPOLYGON (((-70.83204 41.60	
2	0400000US26	26	Michigan	56538.901	MULTIPOLYGON (((-88.68443 48.11	
3	0400000US30	30	Montana	145545.801	POLYGON ((-104.05770 44.99743, -10	
4	0400000US32	32	Nevada	109781.180	POLYGON ((-114.05060 37.00040, -11	

```
alt.Chart(geo_states, title='US states').mark_geoshape().encode(
).properties(
    width=500,
    height=300
).project(
    type='albersUsa'
)
```

```
alt.Chart(...)
```

```
# Add the data
#should i rename 'state' to 'NAME'?
geo_states = geo_states.merge(df, on='NAME')

geo_states.head()
```

	GEO_ID	STATE	NAME	LSAD	CENSUSAREA	geometry
0	0400000US23	23	Maine	30842.923	MULTIPOLYGON (((-67.61976 44.51	
1	0400000US25	25	Massachusetts	7800.058	MULTIPOLYGON (((-70.83204 41.60	

	GEO_ID	STATE	NAME	LSAD	CENSUSAREA	geometry
2	0400000US26	26	Michigan		56538.901	MULTIPOLYGON (((-88.68443 48.11...
3	0400000US30	30	Montana		145545.801	POLYGON ((-104.05770 44.99743, -1...
4	0400000US32	32	Nevada		109781.180	POLYGON ((-114.05060 37.00040, -1...

```

alt.Chart(geo_states, title='PRE-election Hate crime per 100k').mark_geoshape().encode(
    color='avg_hatecrimes_per_100k_fbi',
    tooltip=['NAME', 'avg_hatecrimes_per_100k_fbi']
).properties(
    width=500,
    height=300
).project(
    type='albersUsa'
)

alt.Chart(...)

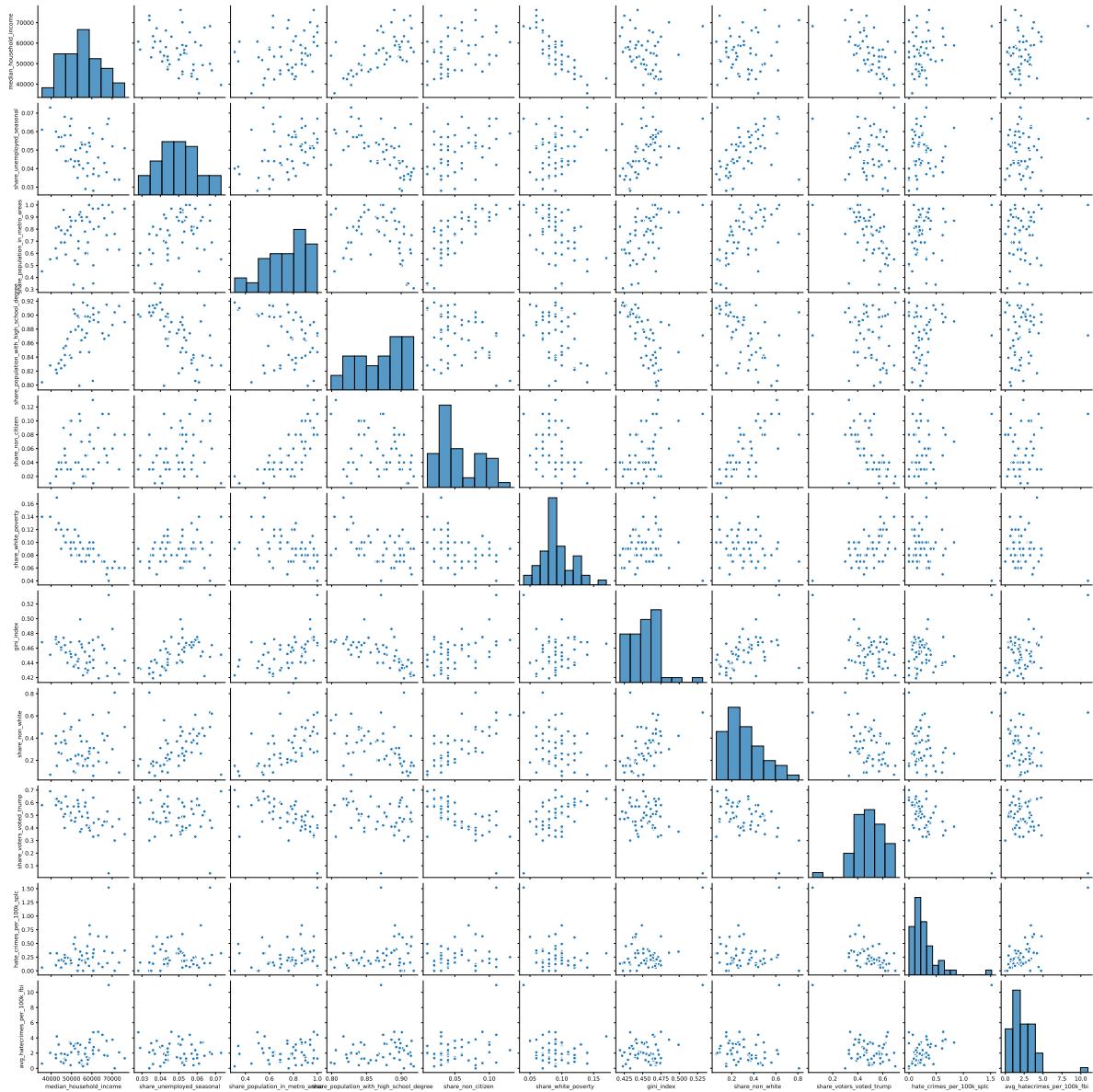
alt.Chart(geo_states, title='POST-election Hate crime per 100k').mark_geoshape().encode(
    color='hate_crimes_per_100k_splc',
    tooltip=['NAME', 'hate_crimes_per_100k_splc']
).properties(
    width=500,
    height=300
).project(
    type='albersUsa'
)

alt.Chart(...)
```

35.5 Exploring data

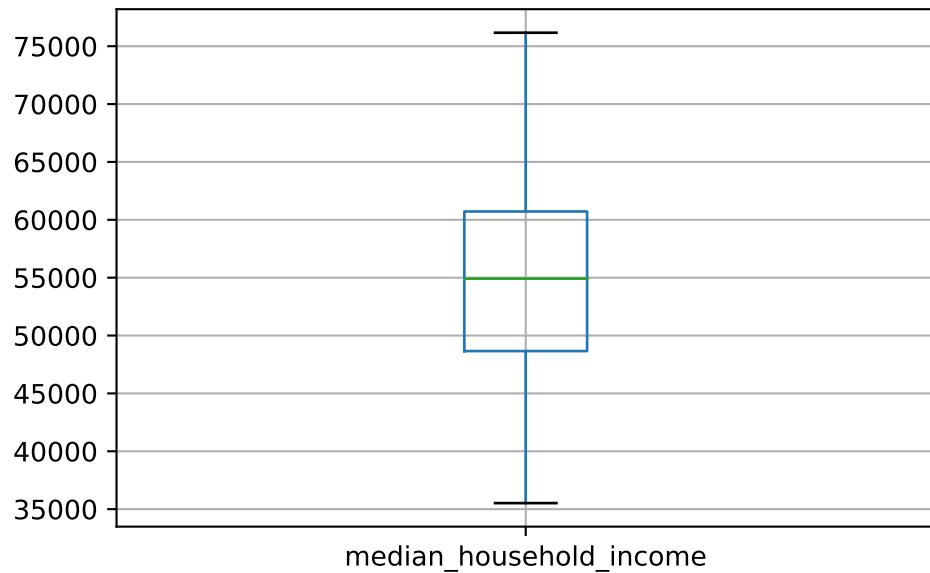
```

import seaborn as sns
sns.pairplot(data = df.iloc[:,1:])
```



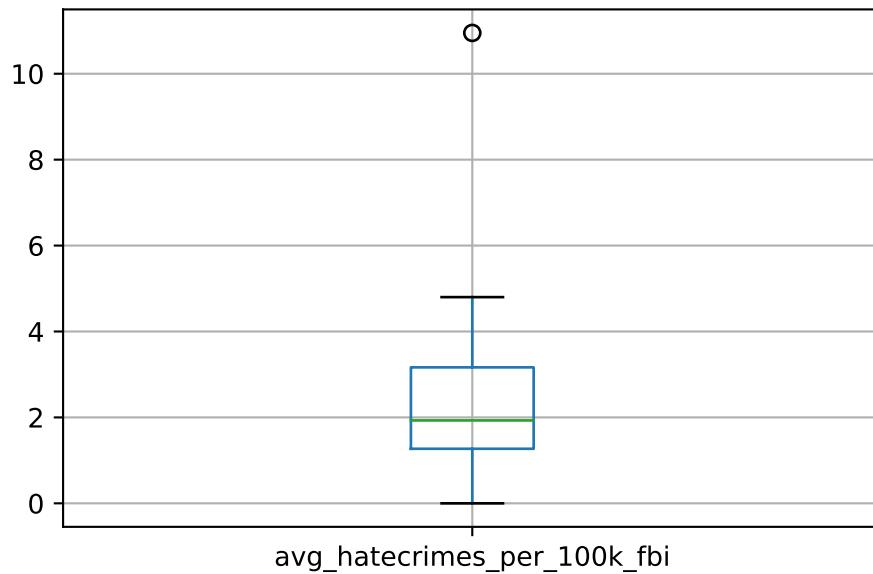
```
df.boxplot(column=['median_household_income'])
```

<Axes: >



```
df.boxplot(column=['avg_hatecrimes_per_100k_fbi'])
```

<Axes: >



We may want to drop columns (remove them). Details are [here](#).

Let us drop Hawaii.

```
df[df.NAME == 'Hawaii']
```

	NAME	median_household_income	share_unemployed_seasonal	share_population_in_metro_areas
11	Hawaii	71223	0.034	0.76

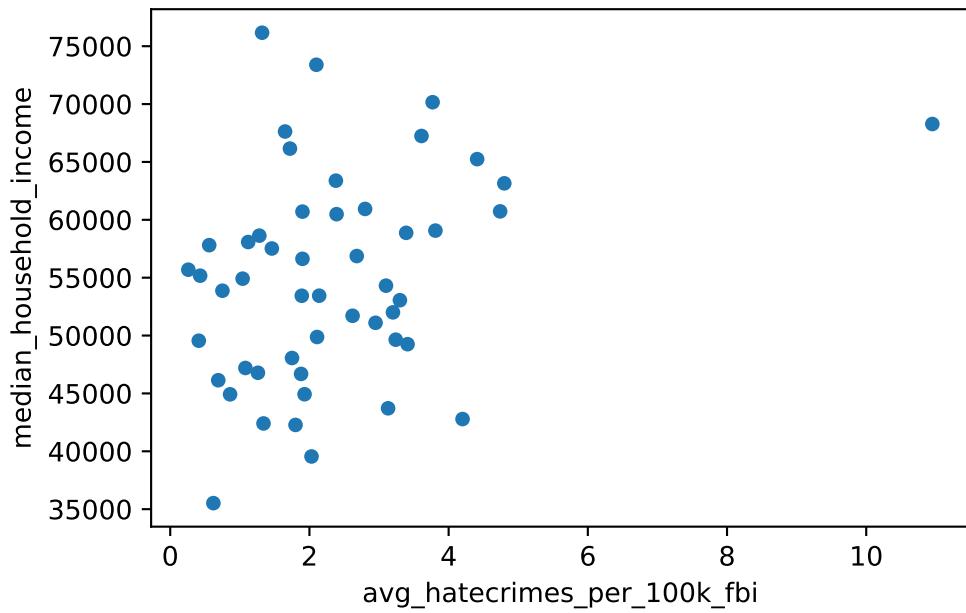
```
df = df.drop(df.index[11])
```

```
df.describe()
```

	median_household_income	share_unemployed_seasonal	share_population_in_metro_areas	sha
count	50.000000	50.000000	50.000000	50.0
mean	54903.620000	0.049880	0.750000	0.80
std	9010.994814	0.010571	0.183425	0.03
min	35521.000000	0.028000	0.310000	0.79
25%	48358.500000	0.042250	0.630000	0.83
50%	54613.000000	0.051000	0.790000	0.87
75%	60652.750000	0.057750	0.897500	0.89
max	76165.000000	0.073000	1.000000	0.91

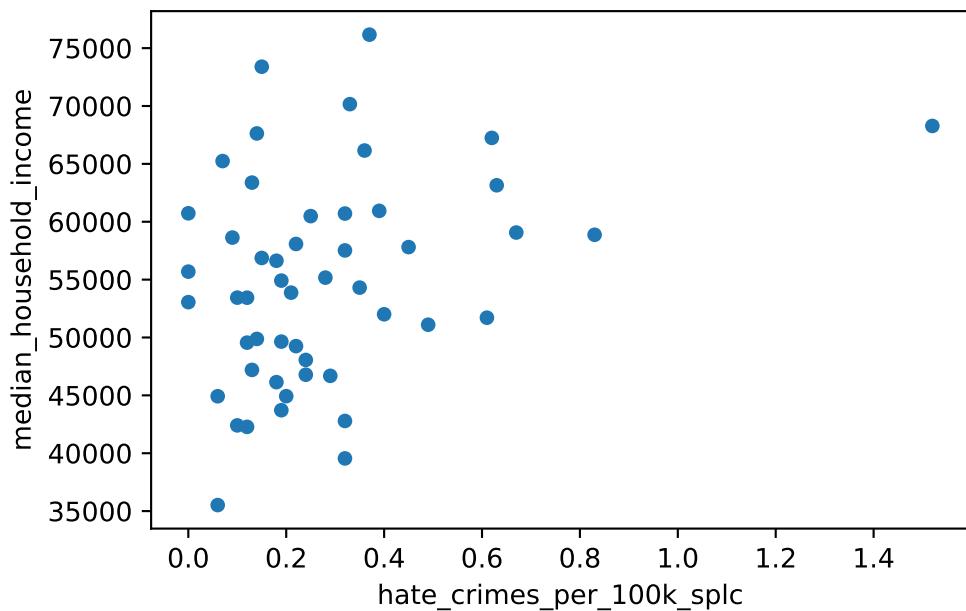
```
df.plot(x = 'avg_hatecrimes_per_100k_fbi', y = 'median_household_income', kind='scatter')
```

```
<Axes: xlabel='avg_hatecrimes_per_100k_fbi', ylabel='median_household_income'>
```



```
df.plot(x = 'hate_crimes_per_100k_splic', y = 'median_household_income', kind='scatter')
```

```
<Axes: xlabel='hate_crimes_per_100k_splic', ylabel='median_household_income'>
```



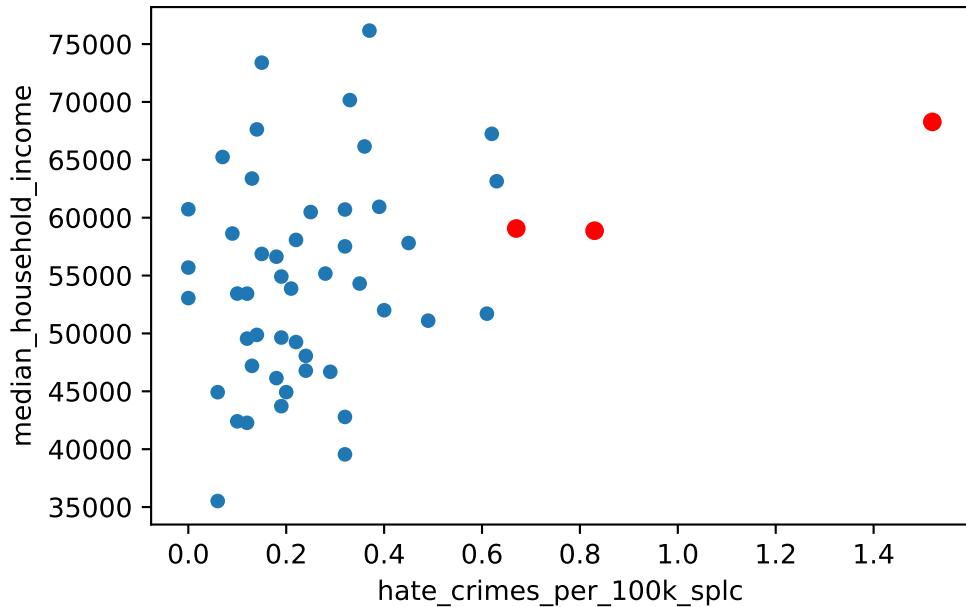
```
df[df.hate_crimes_per_100k_splc > (np.std(df.hate_crimes_per_100k_splc) * 2.5)]
```

	NAME	median_household_income	share_unemployed_seasonal	share_population_in
8	District of Columbia	68277	0.067	1.00
37	Oregon	58875	0.062	0.87
47	Washington	59068	0.052	0.86

```
import matplotlib.pyplot as plt
outliers_df = df[df.hate_crimes_per_100k_splc > (np.std(df.hate_crimes_per_100k_splc) * 2.5)]
df.plot(x = 'hate_crimes_per_100k_splc', y = 'median_household_income', kind='scatter')

plt.scatter(outliers_df.hate_crimes_per_100k_splc, outliers_df.median_household_income, c=
```

```
<matplotlib.collections.PathCollection at 0x169e17610>
```



```
df_pivot = df.pivot_table(index=['NAME'], values=['hate_crimes_per_100k_splc', 'avg_hatecrimes_per_100k_splc', 'share_unemployed_seasonal', 'share_population_in'], fill_value=0)
df_pivot
##sort by values
```

```
#df_pivot = pd.pivot_table(df, index=['state'], columns = ['hate_crimes_per_100k_splc'], fill_value=0)
#df_pivot
#df2 = df_pivot.reindex(df_pivot['hate_crimes_per_100k_splc'].sort_values(by='hate_crimes_per_100k_splc', ascending=False).index)
```

NAME	avg_hatecrimes_per_100k_fbi	hate_crimes_per_100k_splc	median_household_income
Alabama	1.80	0.12	42278
Alaska	1.65	0.14	67629
Arizona	3.41	0.22	49254
Arkansas	0.86	0.06	44922
California	2.39	0.25	60487
Colorado	2.80	0.39	60940
Connecticut	3.77	0.33	70161
Delaware	1.46	0.32	57522
District of Columbia	10.95	1.52	68277
Florida	0.69	0.18	46140
Georgia	0.41	0.12	49555
Idaho	1.89	0.12	53438
Illinois	1.04	0.19	54916
Indiana	1.75	0.24	48060
Iowa	0.56	0.45	57810
Kansas	2.14	0.10	53444
Kentucky	4.20	0.32	42786
Louisiana	1.34	0.10	42406
Maine	2.62	0.61	51710
Maryland	1.32	0.37	76165
Massachusetts	4.80	0.63	63151
Michigan	3.20	0.40	52005
Minnesota	3.61	0.62	67244
Mississippi	0.62	0.06	35521
Missouri	1.90	0.18	56630
Montana	2.95	0.49	51102
Nebraska	2.68	0.15	56870
Nevada	2.11	0.14	49875
New Hampshire	2.10	0.15	73397
New Jersey	4.41	0.07	65243
New Mexico	1.88	0.29	46686
New York	3.10	0.35	54310
North Carolina	1.26	0.24	46784
North Dakota	4.74	0.00	60730
Ohio	3.24	0.19	49644

NAME	avg_hatecrimes_per_100k_fbi	hate_crimes_per_100k_splc	median_household_in
Oklahoma	1.08	0.13	47199
Oregon	3.39	0.83	58875
Pennsylvania	0.43	0.28	55173
Rhode Island	1.28	0.09	58633
South Carolina	1.93	0.20	44929
South Dakota	3.30	0.00	53053
Tennessee	3.13	0.19	43716
Texas	0.75	0.21	53875
Utah	2.38	0.13	63383
Vermont	1.90	0.32	60708
Virginia	1.72	0.36	66155
Washington	3.81	0.67	59068
West Virginia	2.03	0.32	39552
Wisconsin	1.12	0.22	58080
Wyoming	0.26	0.00	55690

```
df_pivot.sort_values(by=['avg_hatecrimes_per_100k_fbi'], ascending=False)
```

NAME	avg_hatecrimes_per_100k_fbi	hate_crimes_per_100k_splc	median_household_in
District of Columbia	10.95	1.52	68277
Massachusetts	4.80	0.63	63151
North Dakota	4.74	0.00	60730
New Jersey	4.41	0.07	65243
Kentucky	4.20	0.32	42786
Washington	3.81	0.67	59068
Connecticut	3.77	0.33	70161
Minnesota	3.61	0.62	67244
Arizona	3.41	0.22	49254
Oregon	3.39	0.83	58875
South Dakota	3.30	0.00	53053
Ohio	3.24	0.19	49644
Michigan	3.20	0.40	52005
Tennessee	3.13	0.19	43716
New York	3.10	0.35	54310
Montana	2.95	0.49	51102
Colorado	2.80	0.39	60940
Nebraska	2.68	0.15	56870

NAME	avg_hatecrimes_per_100k_fbi	hate_crimes_per_100k_splc	median_household_in
Maine	2.62	0.61	51710
California	2.39	0.25	60487
Utah	2.38	0.13	63383
Kansas	2.14	0.10	53444
Nevada	2.11	0.14	49875
New Hampshire	2.10	0.15	73397
West Virginia	2.03	0.32	39552
South Carolina	1.93	0.20	44929
Vermont	1.90	0.32	60708
Missouri	1.90	0.18	56630
Idaho	1.89	0.12	53438
New Mexico	1.88	0.29	46686
Alabama	1.80	0.12	42278
Indiana	1.75	0.24	48060
Virginia	1.72	0.36	66155
Alaska	1.65	0.14	67629
Delaware	1.46	0.32	57522
Louisiana	1.34	0.10	42406
Maryland	1.32	0.37	76165
Rhode Island	1.28	0.09	58633
North Carolina	1.26	0.24	46784
Wisconsin	1.12	0.22	58080
Oklahoma	1.08	0.13	47199
Illinois	1.04	0.19	54916
Arkansas	0.86	0.06	44922
Texas	0.75	0.21	53875
Florida	0.69	0.18	46140
Mississippi	0.62	0.06	35521
Iowa	0.56	0.45	57810
Pennsylvania	0.43	0.28	55173
Georgia	0.41	0.12	49555
Wyoming	0.26	0.00	55690

```
#This is code for standarization
from sklearn import preprocessing
import numpy as np

#Get column names first
#names = df.columns
```

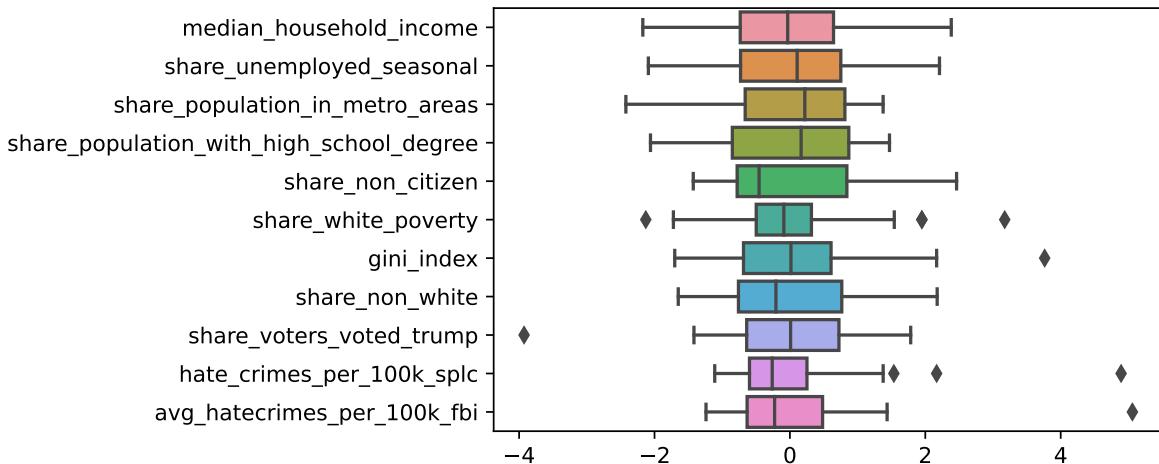
```

#df_stand = df[['median_household_income', 'share_unemployed_seasonal']]
df_stand = df[['median_household_income', 'share_unemployed_seasonal', 'share_population_in_metro_areas', 'share_population_with_high_school_degree', 'share_non_citizen', 'share_non_white', 'share_voters_voted_trump', 'hate_crimes_per_100k_splic', 'avg_hatecrimes_per_100k_fbi']]
names = df_stand.columns
#Create the Scaler object
scaler = preprocessing.StandardScaler()
#Fit your data on the scaler object
df2 = scaler.fit_transform(df_stand)
df2 = pd.DataFrame(df2, columns=names)
df2.tail()

```

	median_household_income	share_unemployed_seasonal	share_population_in_metro_areas	share_population_with_high_school_degree	share_non_citizen	share_non_white	share_voters_voted_trump	hate_crimes_per_100k_splic	avg_hatecrimes_per_100k_fbi
45	1.261305	-0.657461	0.771002						-0.0717
46	0.466836	0.202590	0.605787						0.8478
47	-1.720951	2.209376	-1.101431						-1.1991
48	0.356079	-0.657461	-0.330429						0.8775
49	0.088155	-0.944145	-2.423149						1.4709

```
ax = sns.boxplot(data=df2, orient="h")
```



```
#wanted to remove row with Hawaii (row nr 11) following https://chrisalbon.com/python/data_wrangling/pandas_selecting_data_rows.html
```

```
df2 = df.copy()
```

```

df2
#df2.drop('Hawaii')
#df2.drop(11) #drop Hawaii row
df2.drop(df.index[11])
df2.tail()

```

	NAME	median_household_income	share_unemployed_seasonal	share_population_in_metro
46	Virginia	66155	0.043	0.89
47	Washington	59068	0.052	0.86
48	West Virginia	39552	0.073	0.55
49	Wisconsin	58080	0.043	0.69
50	Wyoming	55690	0.040	0.31

```

import scipy.stats
#instead of running it one by one for every pair of variables, like:
#scipy.stats.pearsonr(st_wine.quality.values, st_wine.alcohol.values)

corrMatrix = df2.corr(numeric_only=True).round(2)
print (corrMatrix)

```

	median_household_income \
median_household_income	1.00
share_unemployed_seasonal	-0.34
share_population_in_metro_areas	0.29
share_population_with_high_school_degree	0.64
share_non_citizen	0.28
share_white_poverty	-0.82
gini_index	-0.15
share_non_white	-0.00
share_voters_voted_trump	-0.57
hate_crimes_per_100k_splc	0.33
avg_hatecrimes_per_100k_fbi	0.32

	share_unemployed_seasonal \
median_household_income	-0.34
share_unemployed_seasonal	1.00
share_population_in_metro_areas	0.37
share_population_with_high_school_degree	-0.61
share_non_citizen	0.31
share_white_poverty	0.19

gini_index	0.53
share_non_white	0.59
share_voters_voted_trump	-0.21
hate_crimes_per_100k_splc	0.18
avg_hatecrimes_per_100k_fbi	0.07
	share_population_in_metro_areas \
median_household_income	0.29
share_unemployed_seasonal	0.37
share_population_in_metro_areas	1.00
share_population_with_high_school_degree	-0.27
share_non_citizen	0.75
share_white_poverty	-0.39
gini_index	0.52
share_non_white	0.60
share_voters_voted_trump	-0.58
hate_crimes_per_100k_splc	0.26
avg_hatecrimes_per_100k_fbi	0.21
	share_population_with_high_school_degree \
median_household_income	0.64
share_unemployed_seasonal	-0.61
share_population_in_metro_areas	-0.27
share_population_with_high_school_degree	1.00
share_non_citizen	-0.30
share_white_poverty	-0.48
gini_index	-0.58
share_non_white	-0.56
share_voters_voted_trump	-0.13
hate_crimes_per_100k_splc	0.21
avg_hatecrimes_per_100k_fbi	0.16
	share_non_citizen \
median_household_income	0.28
share_unemployed_seasonal	0.31
share_population_in_metro_areas	0.75
share_population_with_high_school_degree	-0.30
share_non_citizen	1.00
share_white_poverty	-0.38
gini_index	0.51
share_non_white	0.76
share_voters_voted_trump	-0.62
hate_crimes_per_100k_splc	0.28

avg_hatecrimes_per_100k_fbi	0.30		
median_household_income	-0.82	-0.15	share_white_poverty
share_unemployed_seasonal	0.19	0.53	gini_index \
share_population_in_metro_areas	-0.39	0.52	
share_population_with_high_school_degree	-0.48	-0.58	
share_non_citizen	-0.38	0.51	
share_white_poverty	1.00	0.01	
gini_index	0.01	1.00	
share_non_white	-0.24	0.59	
share_voters_voted_trump	0.54	-0.46	
hate_crimes_per_100k_splc	-0.26	0.38	
avg_hatecrimes_per_100k_fbi	-0.26	0.42	
share_non_white \			
median_household_income	-0.00		
share_unemployed_seasonal	0.59		
share_population_in_metro_areas	0.60		
share_population_with_high_school_degree	-0.56		
share_non_citizen	0.76		
share_white_poverty	-0.24		
gini_index	0.59		
share_non_white	1.00		
share_voters_voted_trump	-0.44		
hate_crimes_per_100k_splc	0.12		
avg_hatecrimes_per_100k_fbi	0.08		
share_voters_voted_trump \			
median_household_income	-0.57		
share_unemployed_seasonal	-0.21		
share_population_in_metro_areas	-0.58		
share_population_with_high_school_degree	-0.13		
share_non_citizen	-0.62		
share_white_poverty	0.54		
gini_index	-0.46		
share_non_white	-0.44		
share_voters_voted_trump	1.00		
hate_crimes_per_100k_splc	-0.69		
avg_hatecrimes_per_100k_fbi	-0.50		
hate_crimes_per_100k_splc \			
median_household_income	0.33		

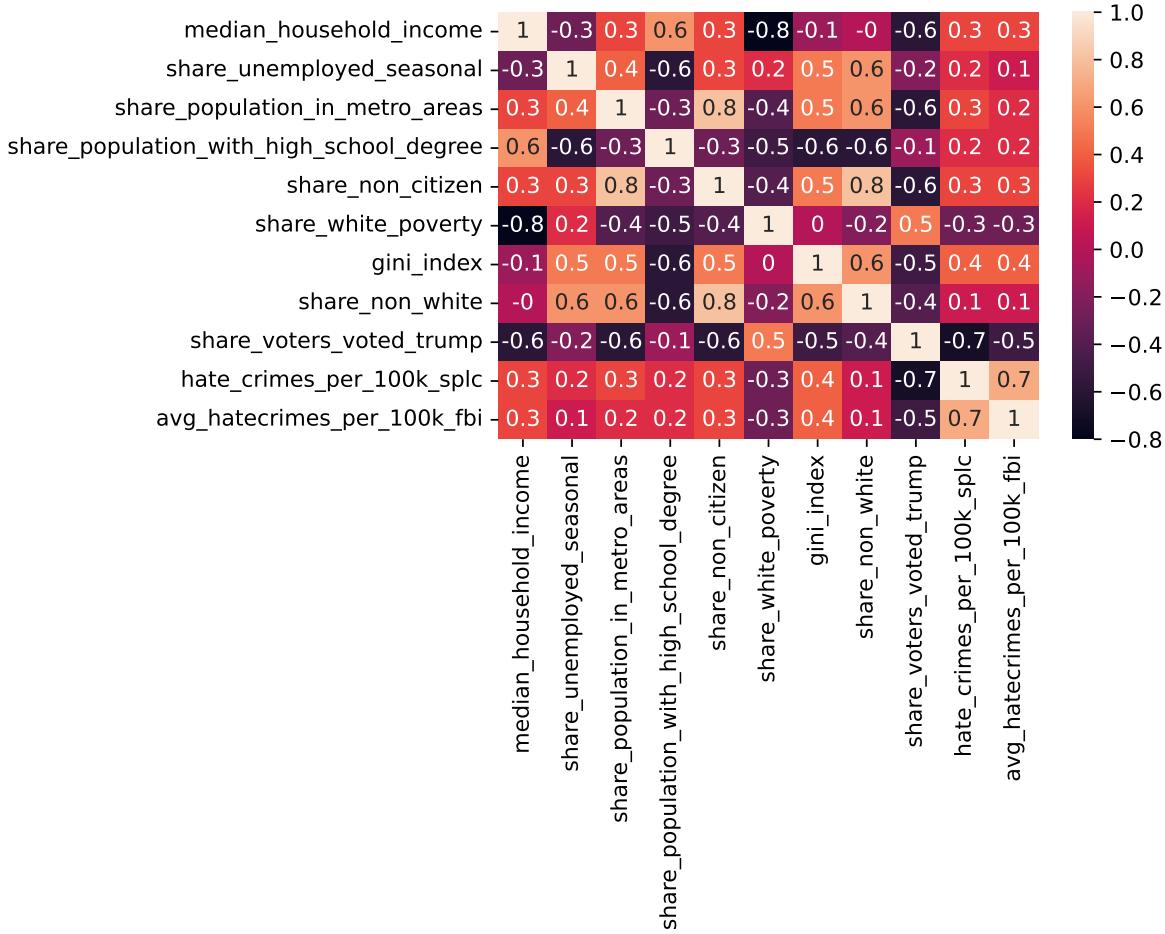
share_unemployed_seasonal	0.18
share_population_in_metro_areas	0.26
share_population_with_high_school_degree	0.21
share_non_citizen	0.28
share_white_poverty	-0.26
gini_index	0.38
share_non_white	0.12
share_voters_voted_trump	-0.69
hate_crimes_per_100k_splc	1.00
avg_hatecrimes_per_100k_fbi	0.68
avg_hatecrimes_per_100k_fbi	0.32
median_household_income	0.07
share_unemployed_seasonal	0.21
share_population_in_metro_areas	0.16
share_population_with_high_school_degree	0.30
share_non_citizen	-0.26
share_white_poverty	0.42
gini_index	0.08
share_non_white	-0.50
share_voters_voted_trump	0.68
hate_crimes_per_100k_splc	1.00

```

import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt

corrMatrix = df2.corr(numeric_only=True).round(1)  #I added here ".round(1)" so that's eas
sn.heatmap(corrMatrix, annot=True)
plt.show()

```



```

import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn import metrics

x = df2[['median_household_income', 'share_population_with_high_school_degree', 'share_voters_voted_trump']]
y = df2[['avg_hatecrimes_per_100k_fbi']]
#what if we change the y variable
#y = df2[['hate_crimes_per_100k_splic']]

est = LinearRegression(fit_intercept = True)
est.fit(x, y)

print("Coefficients:", est.coef_)
print ("Intercept:", est.intercept_)

```

```
model = LinearRegression()
model.fit(x, y)
y_hat = model.predict(x)
print ("MSE:", metrics.mean_squared_error(y, y_hat))
print ("R^2:", metrics.r2_score(y, y_hat))
print ("var:", y.var())
```

```
Coefficients: [[-1.63935828e-05  7.65352737e+00 -7.85302986e+00]]
Intercept: [0.49461694]
MSE: 2.1105276140605045
R^2: 0.26736253642536767
var: avg_hatecrimes_per_100k_fbi      2.939516
dtype: float64
```

36 IM939 - Lab 7 Part 2

The idea of measuring Poverty and Inequality using the case study on “The Statistics of Poverty and Inequality”. (Mary Rouncefield (1995) The Statistics of Poverty and Inequality, Journal of Statistics Education, 3:2, , DOI: 10.1080/10691898.1995.11910491) By looking into 6 variables, you can investigate some major inequalities across the globe.

36.1 Source

WDI Indicators. The data comes from:

<https://databank.worldbank.org/source/world-development-indicators#>

36.1.1 Definitions

Name	Indicator Name	Long definition
birthrate	Birth rate, crude (per 1,000 people)	Crude birth rate indicates the number of live births occurring during the year, per 1,000 population estimated at midyear. Subtracting the crude death rate from the crude birth rate provides the rate of natural increase, which is equal to the rate of population change in the absence of migration.
Neonatal_death	Number of neonatal deaths	Number of neonates dying before reaching 28 days of age.
Lifeexp_male	Life expectancy at birth, female (years)	Life expectancy at birth indicates the number of years a newborn infant would live if prevailing patterns of mortality at the time of its birth were to stay the same throughout its life.
Lifeexp_female	Life expectancy at birth, male (years)	Life expectancy at birth indicates the number of years a newborn infant would live if prevailing patterns of mortality at the time of its birth were to stay the same throughout its life.
GNI	GNI per capita, PPP (current international \$)	This indicator provides per capita values for gross national income (GNI). Formerly GNP) expressed in current international dollars converted by purchasing power parity (PPP) conversion factor. GNI is the sum of value added by all resident producers plus any product taxes (less subsidies) not included in the valuation of output plus net receipts of primary income (compensation of employees and property income) from abroad. PPP conversion factor is a spatial price deflator and currency converter that eliminates the effects of the differences in price levels between countries.
Deathrate	Death rate, crude (per 1,000 people)	Crude death rate indicates the number of deaths occurring during the year, per 1,000 population estimated at midyear. Subtracting the crude death rate from the crude birth rate provides the rate of natural increase, which is equal to the rate of population change in the absence of migration.

Figure 36.1: image-6.png

36.1.2 Questions

The questions that M. Rouncefield asked her students were the following:

1. Is the world's wealth distributed evenly? What countries are outliers?
2. Do people living in different countries have similar life expectancies?
3. Do men and women have similar life expectancies? What is the average difference? What is the minimum difference? What is the maximum difference? In which countries do these occur? What are possible explanations for these differences?
4. Are birth rates related to death rates?
5. How quickly are populations growing?

36.2 Reading the dataset

```
import pandas as pd
df = pd.read_excel('data/WDI_countries_v2.xlsx', sheet_name='Data4')
```

Let's have a look at our dataset

```
df.head()
```

	Country Code	birthrate	Deathrate	GNI	Lifeexp_female	Lifeexp_male	Neonatal_death
0	AFG	32.487	6.423	2260.0	66.026	63.047	44503.0
1	ALB	11.780	7.898	13820.0	80.167	76.816	243.0
2	DZA	24.282	4.716	11450.0	77.938	75.494	16407.0
3	AND	7.200	4.400	NaN	NaN	NaN	1.0
4	AGO	40.729	8.190	6550.0	63.666	58.064	35489.0

36.2.1 Missing values

Let's check if we have any missing data

```
df.info()
df.isna().sum()
```

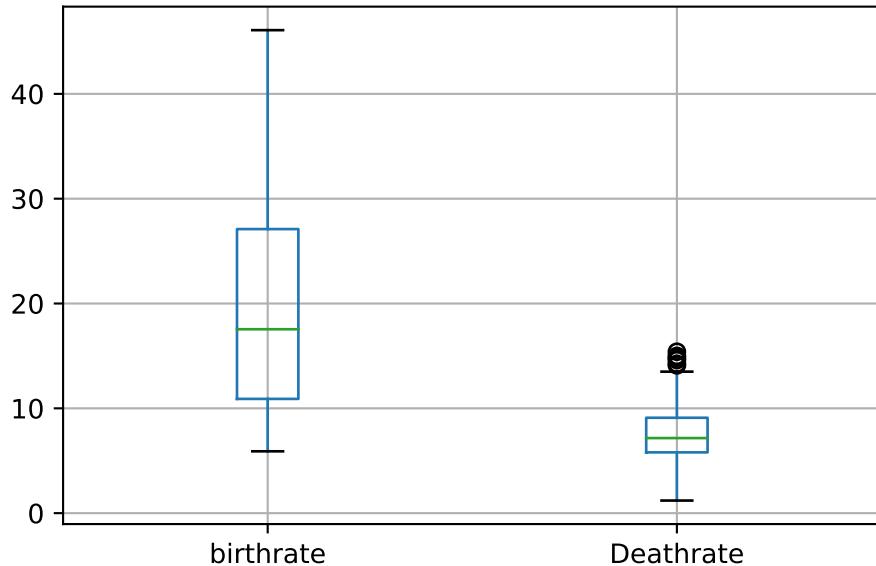
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 216 entries, 0 to 215
Data columns (total 7 columns):
```

```
#   Column      Non-Null Count   Dtype  
---  --  
0   Country Code    216 non-null   object  
1   birthrate      205 non-null   float64 
2   Deathrate      205 non-null   float64 
3   GNI            187 non-null   float64 
4   Lifeexp_female 198 non-null   float64 
5   Lifeexp_male   198 non-null   float64 
6   Neonatal_death 193 non-null   float64 
dtypes: float64(6), object(1)  
memory usage: 11.9+ KB
```

```
Country Code      0
birthrate        11
Deathrate        11
GNI              29
Lifeexp_female   18
Lifeexp_male    18
Neonatal_death  23
dtype: int64
```

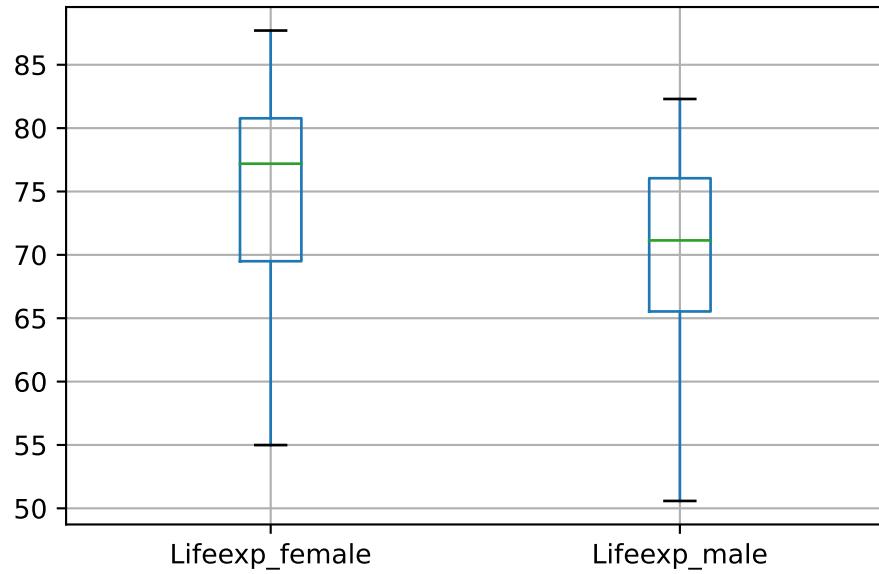
```
df.boxplot(column=['birthrate', 'Deathrate'])
```

```
<Axes: >
```



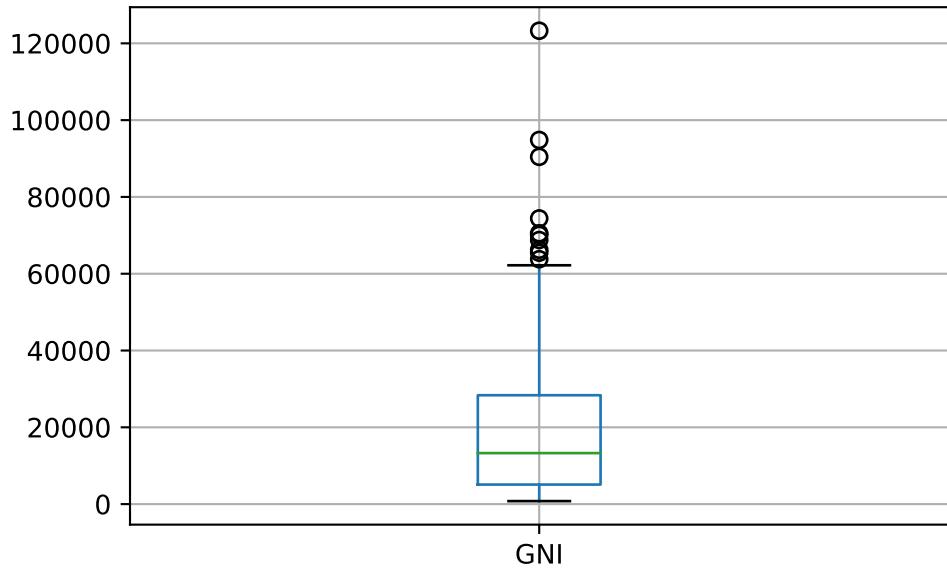
```
df.boxplot(column=['Lifeexp_female', 'Lifeexp_male'])
```

<Axes: >



```
df.boxplot(column=['GNI'])
```

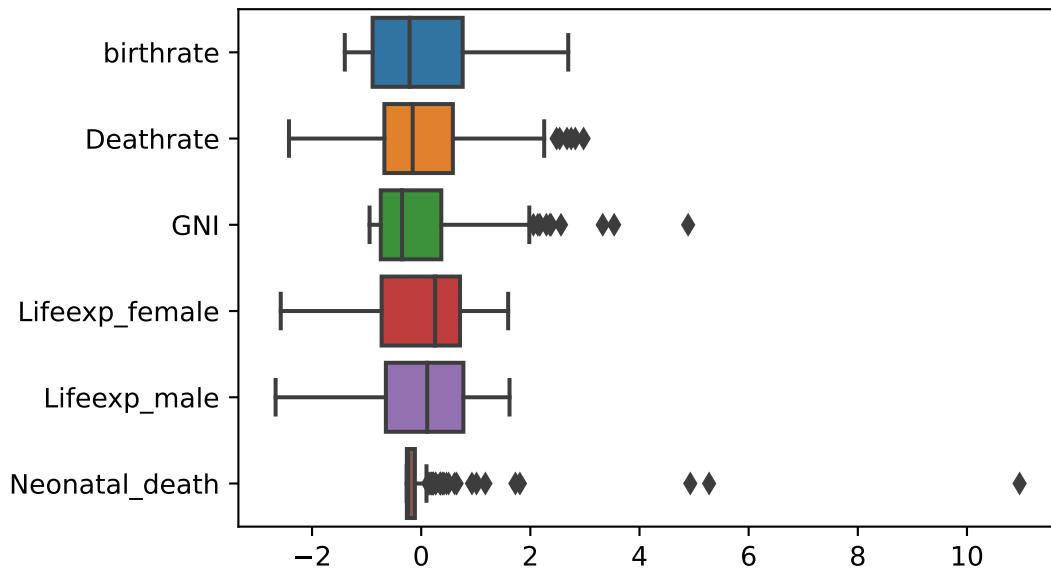
<Axes: >



```
#This is code for standarization
from sklearn import preprocessing
import numpy as np
import seaborn as sns

#Get column names first
#names = df.columns
df_stand = df[['birthrate', 'Deathrate', 'GNI', 'Lifeexp_female', 'Lifeexp_male', 'Neonatal_mortality_rate']]
names = df_stand.columns
#Create the Scaler object
scaler = preprocessing.StandardScaler()
#Fit your data on the scaler object
df2 = scaler.fit_transform(df_stand)
df2 = pd.DataFrame(df2, columns=names)
df2.tail()

ax = sns.boxplot(data=df2, orient="h")
```

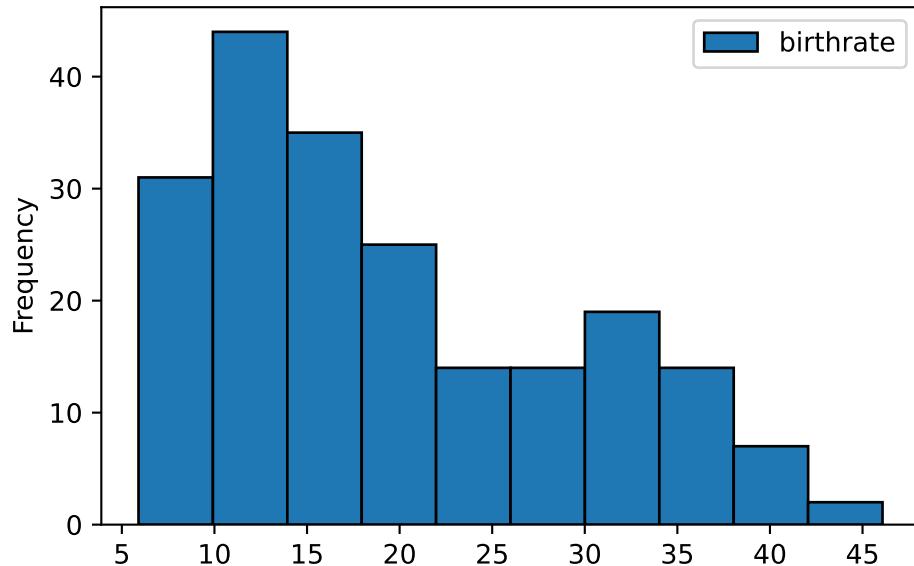


```
df.describe()
```

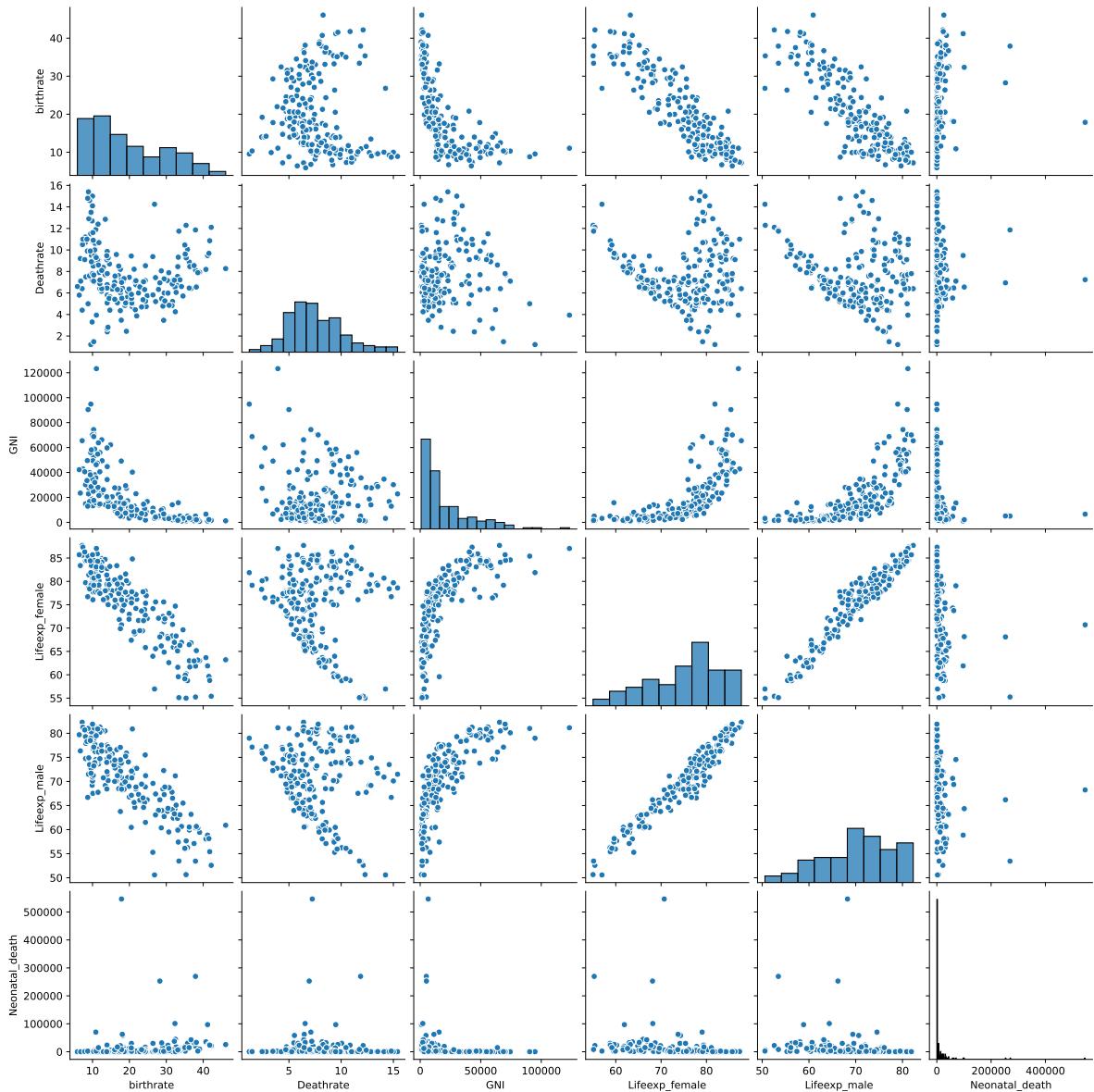
	birthrate	Deathrate	GNI	Lifeexp_female	Lifeexp_male	Neonatal_death
count	205.000000	205.000000	187.000000	198.000000	198.000000	193.000000
mean	19.637580	7.573941	20630.427807	75.193288	70.323854	12948.031088
std	9.839573	2.636414	21044.240160	7.870933	7.419214	48782.770706
min	5.900000	1.202000	780.000000	54.991000	50.582000	0.000000
25%	10.900000	5.800000	5090.000000	69.497250	65.533500	163.000000
50%	17.545000	7.163000	13280.000000	77.193000	71.140500	1288.000000
75%	27.100000	9.100000	28360.000000	80.776500	76.047500	7316.000000
max	46.079000	15.400000	123290.000000	87.700000	82.300000	546427.000000

```
df[['birthrate']].plot(kind='hist', ec='black')
```

```
<Axes: ylabel='Frequency'>
```

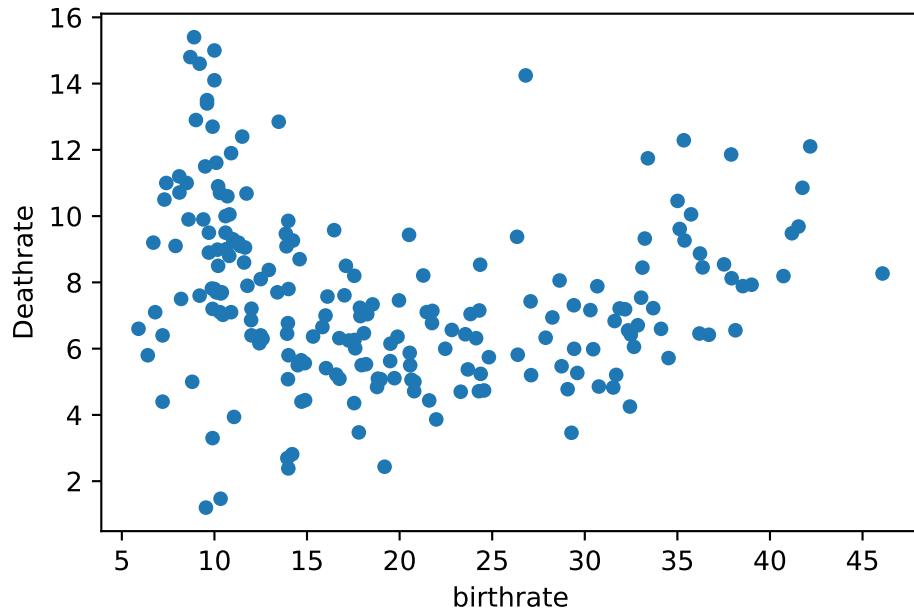


```
import seaborn as sns
sns.pairplot(data = df.iloc[:,1:])
```



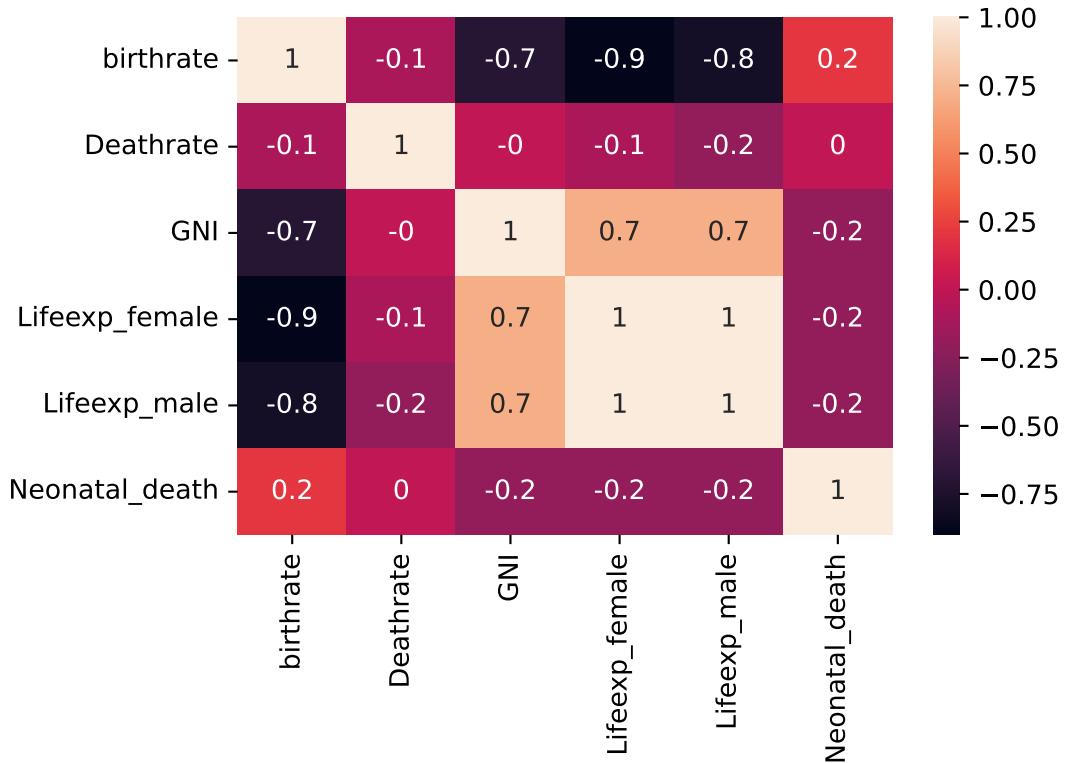
```
df.plot(x = 'birthrate', y = 'Deathrate', kind='scatter')
```

```
<Axes: xlabel='birthrate', ylabel='Deathrate'>
```



```
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt

corrMatrix = df.corr(numeric_only=True).round(1)  #I added here ".round(1)" so that's easier
sn.heatmap(corrMatrix, annot=True)
plt.show()
```



36.3 How quickly are populations growing?

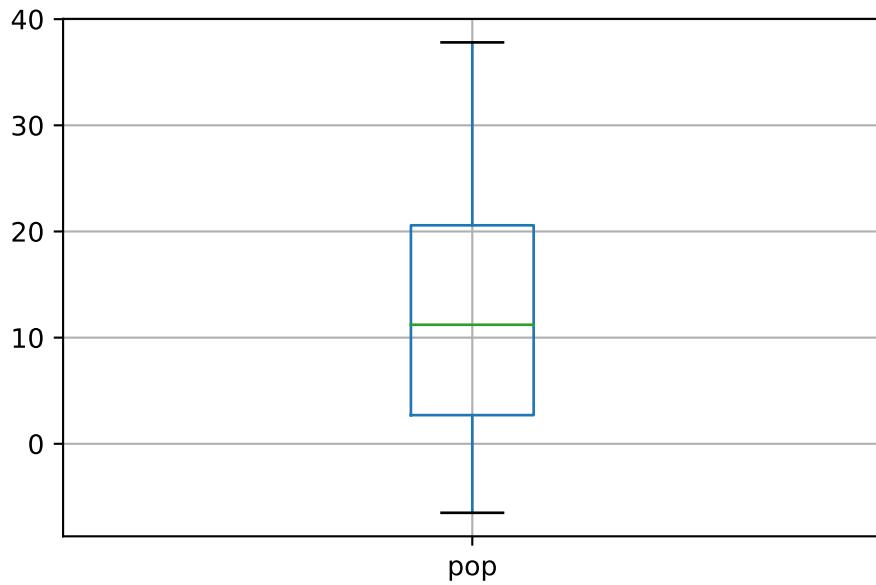
This question can be investigated by calculating birth rate minus death rate. Results range from -?? (a decreasing population) to +?? (an increasing population). The mean is around ??, but what does it signify?

```
df['pop'] = df['birthrate'] - df['Deathrate']
df.head()
```

	Country	Code	birthrate	Deathrate	GNI	Lifeexp_female	Lifeexp_male	Neonatal_death	pop
0	AFG		32.487	6.423	2260.0	66.026	63.047	44503.0	26.0
1	ALB		11.780	7.898	13820.0	80.167	76.816	243.0	3.88
2	DZA		24.282	4.716	11450.0	77.938	75.494	16407.0	19.5
3	AND		7.200	4.400	NaN	NaN	NaN	1.0	2.80
4	AGO		40.729	8.190	6550.0	63.666	58.064	35489.0	32.5

```
df.boxplot(column=['pop'])
```

<Axes: >



References

A Setup & Usage

This repository uses the following technologies:

- conda for creating and managing virtual environments
- jupyter notebooks for course materials combining rich-text and code
- [quarto](#) to generate the handbook

A.1 Setting up the environment

Virtual environments are a way to install all the dependencies (and their right version) required for a certain project by isolating python and libraries' specific versions. Every person who recreates the virtual environment will be using the same packages and versions, reducing errors and increasing reproducibility.

This project, uses a `conda` environment called `IM939`, which will install all the packages and versions (as well as their dependencies) as defined in the file `environment.yml` at the root of this project.

A.1.1 Installing Anaconda

You will need to install [Anaconda distribution](#) your machine if it is not already installed. You can run the following command to see if it is already present in your system:

```
conda --help
```

If you get a `command not found` error, you will need to install Anaconda following [these instructions on their website](#)).

A.1.2 Creating the virtual environment

Note

You only need to do this once in the same machine. After the virtual environment is created we can always update it to follow any change in the file `environment.yml`

To recreate the virtual environment from `environment.yml`, run the following command:

```
conda env create -f environment.yml
```

or, if we want to install the environment within the project:

```
conda env create --prefix env -f environment.yml
```

A.1.3 Activating the virtual environment

Once the environment has been created, it needs to be activated by typing the following command

```
conda activate IM939
```

or, if it is stored in `env/` folder:

```
conda activate env/
```

Once per session

You will need to activate the environment every time you open your editor anew.

Deactivate virtual environment:

If you want, you can always deactivate your environment (and, actually open the default one, called `base`) by running:

```
conda deactivate
```

Update virtual environment from `environment.yml`:

```
conda env update -f environment.yml
```

Freeze used dependencies into a file

We can create a file (in this case `environment.yml`) containing the exact libraries and versions used in the current environment. This can be useful to update the versions used in the environment in the future.

```
conda env export > environment.yml
```

A.2 Recreating the handbook



Tip

Quarto has extensive documentation at their website (specifically in this page about authoring document and this other on managing books).

The workflow can be summarised as follows (detailed instructions below):

1. Create new content or edit existing one
 1. Create a `.md`, `.iypnb` or `.qmd` file and put it in the `/content/` folder
 2. Add an entry to the table of contents in `_quarto.yml` (more info in quarto)
2. Edit existing content in `/content/` folder
3. Render book locally to see changes
4. Commit & Push changes to the source documents in `/content/` folder

```
git commit -a -m "My fancy message"  
git push origin main
```

5. Publish book online

A.2.1 Rendering the book locally:

Rendering the book will convert jupyternotebooks, qmd files or md files listed in `_quarto.yml`'s table of contents into a book, applying the styles and configurations from `_quarto.yml`. Do this to preview how your changes would look like as a book. From the repo's root, run:

```
quarto render
```

⚠ First time render

On its first run, this command will take several minutes to process. This is because quarto will run and execute the computations in every cell within every jupyter notebook, some of which are really time consuming. The good news, is that quarto will create a cached version of it (stored in the `/_freeze/` folder) , which means that further runs of `quarto render` will not need to execute the cells again (unless the original jupyter notebook is changed or the corresponding folder is deleted).

If you want the cache to be regenerated:

```
quarto render --cache-refresh
```

A.2.2 Publishing book to github pages

This book is published using GitHub pages, and assumes that your rendered book will be located on a dedicated branch called `gh-pages` which needs to be created in the repo if not present already. Also, the repository needs to be configured as to use that branch's root for publishing a GitHubPage.

Once `gh-pages` branch has been created, run the following command:

```
quarto publish gh-pages
```

This command will render the book in the branch `gh-pages` and will push it to the corresponding branch in our repo and then checking out again to the previous branch (usually, `main`). More info about it here: <https://quarto.org/docs/publishing/github-pages.html>

ℹ Other useful resources

- Notebook embedding: <https://quarto.org/docs/authoring/notebook-embed.html>

B Working with files and directories

This is a placeholder

```
# SETTING THE WORKING DIRECTORY: if you have not downloaded both the .ipynb file and the D
# location, then put the path to the folder containing all of your data files (necessary f
# the os.chdir() function as a STRING
os.chdir(os.path.join(os.getcwd(), 'data'))
# READ COMMENT ABOVE or Download the Data folder as well into the same folder as this note
```