Inf2 - Foundations of Data Science 2021-22

October 24, 2023

Inf2-FDS Coursework 1 - Data wrangling and visualisation

Released: Monday, 25 October 2021

Submission deadline: **Friday, 5 November, 16:00 UK time**

Late submission rules

This coursework uses the Informatics Late Submission of Coursework Rule 3 with a **maximum 6 day** extension:

> Extensions, Extra Time Adjustment (ETA) for Extra Time and for Extra Time for Proof Reader/Interpreter are permitted, **but cannot be combined**. The maximum extension is up to 6 days, or fewer if specified.
>
> **Penalty:** If assessed coursework is submitted late without an approved ETA extension, it will be recorded as late and a penalty of 5% per calendar day will be applied for up to the specified number of calendar days (≤ 6), after which a mark of zero will be given.
>
> - For electronic submissions, the last version that has been submitted by the deadline will be the one that is marked (late submission will only be accepted if no submission in time has been made).
> - If a student with an extension or either type of ETA submits late beyond the specified extended deadline a mark of zero will be given.

It is very important that you read and follow the instructions below to the letter: you will be deducted marks for not adhering to the advice below.

Good Scholarly Practice

Please remember the University requirement as regards all assessed work for credit. Details about this can be found at: `http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`

Specifically, this coursework **must be your own work**. We want you to be able to discuss the class material with each other, but the coursework you submit must be your own work. You are free to form study groups and discuss the concepts related to, and the high-level approach to the coursework. **You may never share code or share write-ups**. It is also not permitted to discuss this coursework on Piazza. The only exception is that if you believe there is an error in the coursework, you may ask a private question to the instructors, and if we feel that the issue is justified, we will send out an announcement.

Assessment information and criteria

- This assignment accounts for 20% of the grade for this course.
- It is graded based on the PDF export from a Jupyter notebook (see **General instructions** below), which you are to submit via Gradescope (see **Assessment/Coursework 1 - Data wrangling and visualisation** folder in Learn).
- The assignment is marked out of 100 and the number of points is indicated by each question.
- We will assess your work on the following criteria:
  - functional code that performs the computations asked for, as measured by verifying some of the numeric outputs from your processing and reading your code if there is doubt about what you have done
  - the quality of the visualisations, measured against the *Visualisation principles and guidance* handout in the S1 Week 5 workshop (PDF available in S1 Week 5 folder in Learn)
  - the quality of your textual comments - as measured by how accurate, complete and insightful they are.

General instructions

- Read the instructions carefully, answering what is required and only that.
- Fill in your answers in the cells indicated. You may delete text like "Your answer to Q1.2 goes here". **Do not edit or delete any other cells.**
- Keep your answers brief and concise.
- For answers involving visualisations, make sure to label them clearly and provide legends where necessary.
- For answers involving numerical values, use correct units where appropriate and format floating point values to a reasonable number of decimal places.
- Once you have finished a question, use a Jupyter notebook server to export your PDF, by selecting **File->Download as->PDF via LaTeX (.pdf)**.
- Check this PDF document looks as you expect it to. If changes are needed, update the notebook and export again.
- Once you have finished all the questions, submit the final PDF using the submission instructions in the **Assessment->Coursework 1 - Data wrangling and visualisation** folder in Learn. **Please allow enough time to upload your PDF before the deadline.**

Coursework questions

This coursework consists of five questions, divided into three parts (A, B and C). Complete all questions.

We ask for multiple types of responses:

- Numeric responses, which we will use to verify that the processing has been done correctly

- Visualisations, which will be assessed using the parts of the *Visualisation principles and guidance* PDF that apply to the visualisation in question
- Comments on your findings, which may ask you to describe, explain or interpret your results, and which will be assessed on how accurate, complete and insightful they are.

Throughout the coursework, a decade refers to ten years starting from a year divisible by ten, e.g. 1850-1859 (1850s), 1860-1869 (1860s), etc.

In order to understand the meaning the datasets, you may need to follow the links citing the data.

Read through all of a question before starting, as some parts build on each other.

Good luck - we hope you enjoy it!

[5]:
```
# Imports - run this cell first, and add your own imports here.
# You can use any package you want, but we suggest you stick to ones used in␣
 ↪the labs
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams['figure.dpi'] = 300 # Make figures have reasonable␣
 ↪resolution when exporting
```

Part A - Identifying and correcting bad visualisation practices

Designing a good visualisation can be time-consuming, but is important for clear communication. In this part we will ask you to identify and correct bad visualisation practices. (Note that we'd like you to continue using good visualisation practices in other parts of this coursework too!)

Question 1 (15 points)

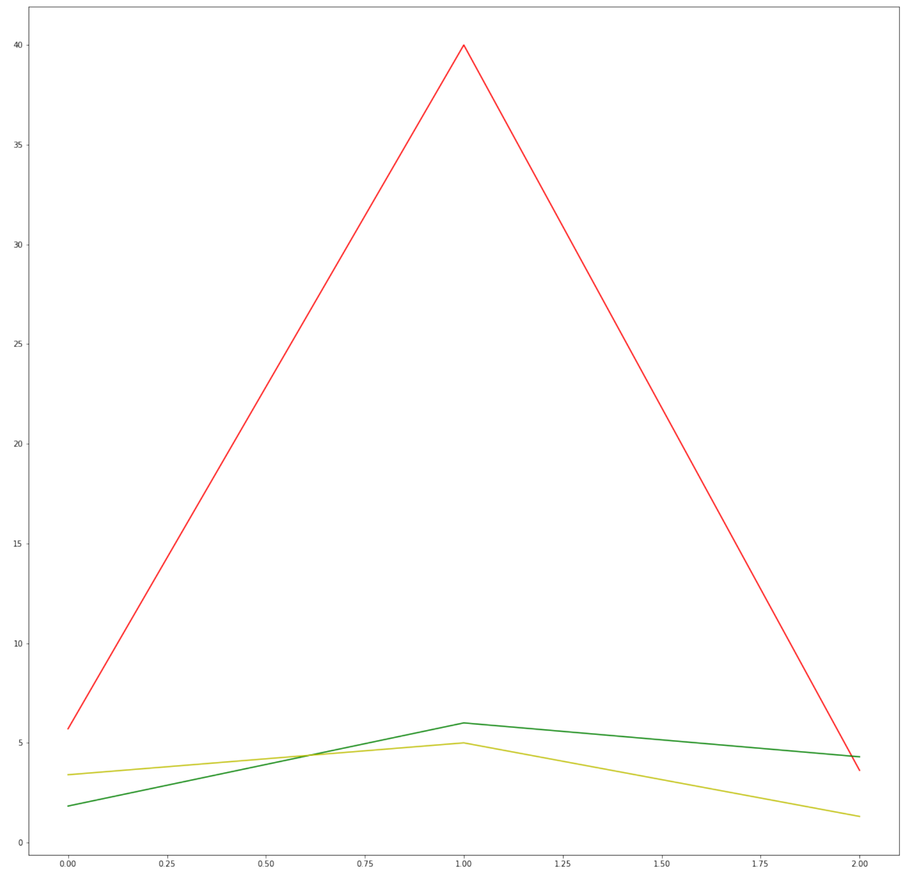
The code in the cell below sets up a small data frame that relates cities, the population density of the city, the number of universities in the city and the percentage of the population that commutes by bicycle in the city.

[6]:
```
# This dataframe holds the data. The columns explain the values.
data = pd.DataFrame(columns=['City',
                             'pop. density [k/km^2]',
                             '# universities',
                             '% commuting by bike'])
data.loc[0] = ['London', 5.701, 40, 3.62]
data.loc[1] = ['Edinburgh', 1.830, 6, 4.3]
data.loc[2] = ['Glasgow', 3.400, 5, 1.31]

data
```

[6]:
```
        City  pop. density [k/km^2] # universities  % commuting by bike
0     London                  5.701             40                 3.62
1  Edinburgh                  1.830              6                 4.30
```

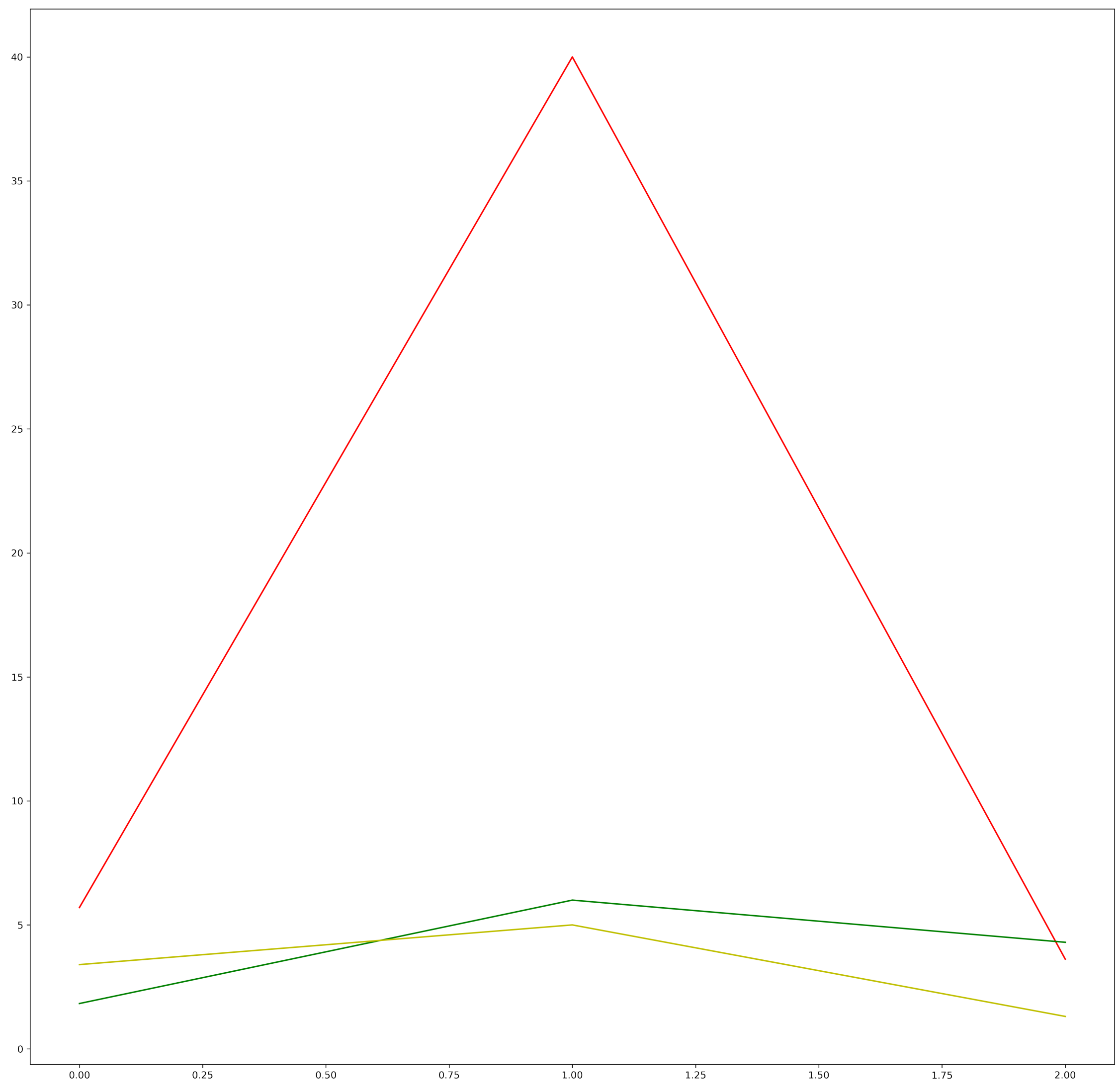
```
2    Glasgow                      3.400              5                    1.31
```

The next cell contains code for a plot that is an example of bad visualisation practice. It shows the population density in 1000s per km^2 ($x = 0$), the number of universities ($x = 1$) and the percentage of people commuting by bicycle ($x = 2$) London (red), Edinburgh (green) and Glasgow (yellow). The plot is also given here for reference.



```
[7]: # This is the figure for you to improve on.
     fig, ax1 = plt.subplots(figsize=(20, 20))
     ax1.plot(list(data.iloc[0])[1:], color='r')
     ax1.plot(list(data.iloc[1])[1:], color='g')
     ax1.plot(list(data.iloc[2])[1:], color='y')
```

```
[7]: [<matplotlib.lines.Line2D at 0x7f93c9ef2dc0>]
```



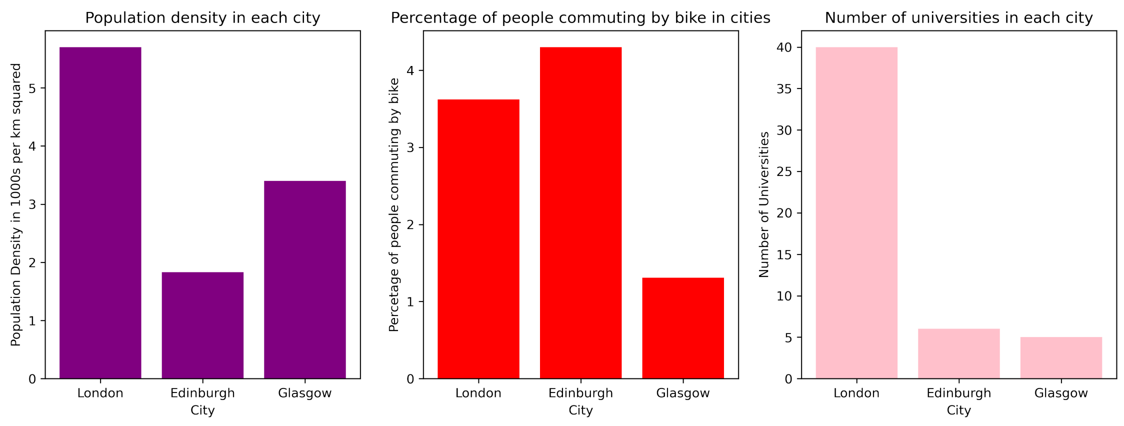
0
5
10
15
20
25
30
35
40
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Question 1.1 (8 points)

Run the code above to produce the plot. Now make an improved version of the plot, so someone presented with it could easily interpret it. There are at least five changes you can make to improve it. Remember that when you export this notebook to a PDF, it will have a text width of 6 inches.

```
[8]: import matplotlib.pyplot as plt

     y=data['pop. density [k/km^2]']
     y1=data['% commuting by bike']
     y2=data['# universities']

     fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize=(15, 5))

     ax1.bar(data['City'], y, color = 'purple')
     ax2.bar(data['City'], y1, color = 'red')
     ax3.bar(data['City'], y2, color = 'pink')

     ax1.set_title('Population density in each city')
     ax1.set_ylabel('Population Density in 1000s per km squared')
     ax1.set_xlabel('City')

     ax2.set_title('Percentage of people commuting by bike in cities')
     ax2.set_ylabel('Percetage of people commuting by bike')
     ax2.set_xlabel('City')

     ax3.set_title('Number of universities in each city')
     ax3.set_ylabel('Number of Universities')
     ax3.set_xlabel('City')
```

```
[8]: Text(0.5, 0, 'City')
```



Question 1.2 (5 points)

List the changes you have made. Explain briefly why each change is an improvement.

Your answer to Q1.2 goes here

1. Start axis at (0,0), this makes the data easier to compare. Otherwise this data would be much harder to compare.
2. I have added axis titles also which make the plot readable.
3. Title over each graph to make it clear what each subplot is displaying.
4. I have also split the one graph into three subplots so that the data is much more readable.
5. I have different colours for each variable so that each graph stands out from eachother.

Question 1.3 (2 points)

Describe the trends that your improved plot shows.

Your answer to Q1.3 goes here

The city London with the highest population density also has highest number of universities by a significant number, although this is not a consistent trend. London also has quite a high percentage of people, just slightly less than the highest value which was Edinburgh. The city Edinburgh with the lowest population density also has the highest percentage of people who commute by bike. Glasgow, which has the lowest number of universities and lowest percentage of people commuting by bike, surprisingly doesn't have the lowest population density.

Part B - Cleaning and exploring UK weather data

Questions 2 and 3 consider historical weather data in the UK. The data comes from https://www.metoffice.gov.uk/research/climate/maps-and-data/historic-station-data, but has been processed to get you started. The data consists of separate files for each station, with each data file having a header with coordinates and information on how to interpret the data. We use only the columns giving the year, rainfall and sunshine. Some of the data is missing. Some values are marked to denote them being special in some way, as described in the header of each file.

Question 2 (25 points)

In question 2 we ask you to read in the weather data, explore it, and look for trends. To make trends clear, we will ask you to average across all sites. To make trends even clearer, we will ask you to smooth out the localised highs and lows of individual years by averaging over decades. We also want you to consider whether there are any limitations of the data that should be considered.

The data for this question is in the folder `weather_sites`, and will need some cleaning.

Questions 2.1 and 2.2 can be solved together, so make sure you read through both before starting.

Question 2.1 (10 points)

Read in and clean the data in the folder `weather_sites`. Then compute: - The mean rainfall in mm/month measured at UK weather stations for the decades 1850s and 1990s (one number per decade). Give the answers to 2 decimal places. - The mean sunlight in hours/month measured at UK weather stations for the years 1899 and 1999 (one number per year). Give the answers to 2 decimal places. - The number of data rows you are using (the number of rows in the Pandas DataFrame you used to compute the mean rainfall and mean sunlight).

To make your answers clear, please either write out the answer in the cell for "your written answer" below the code cell, or make sure that your code produces very clear output.

Hint: check the data type of each column to verify the processing. Include all values, also those marked as provisional, estimates (*), or special in some way ($ and #).

Hint: use `.map(lambda xx: str(xx))` on a data series to convert all entries to strings, `.rstrip(cc)` on strings to remove the character `cc` from the right of a string, and `.astype('float')` to change the data type back to float.

```
[9]: # Import packages
import os
import pandas as pd
import numpy as np
import seaborn as sns
import glob
import statistics

#list of filepath csv names
filepaths = ['weather_sites/aberporthdata.csv','weather_sites/armaghdata.
 ↪csv','weather_sites/ballypatrickdata.csv','weather_sites/bradforddata.
 ↪csv','weather_sites/braemardata.csv','weather_sites/cambornedata.
 ↪csv','weather_sites/cambridgedata.csv','weather_sites/cardiffdata.
 ↪csv','weather_sites/chivenordata.csv','weather_sites/cwmystwythdata.
 ↪csv','weather_sites/dunstaffnagedata.csv','weather_sites/durhamdata.
 ↪csv','weather_sites/eastbournedata.csv','weather_sites/eskdalemuirdata.
 ↪csv','weather_sites/heathrowdata.csv','weather_sites/hurndata.
 ↪csv','weather_sites/lerwickdata.csv','weather_sites/leucharsdata.
 ↪csv','weather_sites/lowestoftdata.csv','weather_sites/manstondata.
 ↪csv','weather_sites/nairndata.csv','weather_sites/newtonriggdata.
 ↪csv','weather_sites/oxforddata.csv','weather_sites/paisleydata.
 ↪csv','weather_sites/ringwaydata.csv','weather_sites/rossonwyedata.
 ↪csv','weather_sites/shawburydata.csv','weather_sites/sheffielddata.
 ↪csv','weather_sites/southamptondata.csv','weather_sites/stornowaydata.
 ↪csv','weather_sites/suttonboningtondata.csv','weather_sites/tireedata.
 ↪csv','weather_sites/valleydata.csv','weather_sites/waddingtondata.
 ↪csv','weather_sites/whitbydata.csv','weather_sites/wickairportdata.csv',␣
 ↪'weather_sites/yeoviltondata.csv']

weather_df = pd.concat(map(lambda p: pd.read_csv(p, header = 6), filepaths))
```

```
weather_df.drop('comment', inplace=True, axis=1)
weather_df.drop('af', inplace=True, axis=1)

weather_df.rain = weather_df['rain'].str.replace('#','')
weather_df.sun = weather_df['sun'].str.replace('#','')
weather_df.rain = weather_df['rain'].str.replace('*','')
weather_df.sun = weather_df['sun'].str.replace('*','')
weather_df.sun = weather_df['sun'].str.replace('$','')

#print(weather_df)

discard = ['---']

sun_df = weather_df[~weather_df.sun.str.contains('|'.join(discard),  na=False)]
#sun_df = sun_df['sun']
sun_df.dropna(subset = ['sun'], inplace=True)
sun_df['sun'] = sun_df.sun.astype('float')

rain_df = weather_df[~weather_df.rain.str.contains('|'.join(discard), ␣
 ↪na=False)]
#rain_df = rain_df['rain']
rain_df.dropna(subset = ['rain'], inplace=True)
rain_df['rain'] = rain_df.rain.astype('float')

#A.1

a1_rain_df = rain_df[(rain_df['yyyy']>= 1850) & (rain_df['yyyy'] < 1860)]
a1_rain_mean = round(a1_rain_df.rain.mean(), 2)

print("Mean rainfall in 1850s:", a1_rain_mean)

#A.2

a2_rain_df = rain_df[(rain_df['yyyy']>= 1990) & (rain_df['yyyy'] < 2000)]
a2_rain_mean = round(a2_rain_df.rain.mean(), 2)

print("Mean rainfall in 1990s:", a2_rain_mean)

#B.1

b1_sun_df = sun_df[(sun_df['yyyy'] == 1899)]
b1_sun_mean = round(b1_sun_df.sun.mean(), 2)

print("Mean sunlight in 1899:", b1_sun_mean)

#B.2
```

```python
b2_sun_df = sun_df[(sun_df['yyyy'] == 1999)]
b2_sun_mean = round(b2_sun_df.sun.mean(), 2)

print("Mean sunlight in 1999:", b2_sun_mean)
```

```
Mean rainfall in 1850s: 58.65
Mean rainfall in 1990s: 76.11
Mean sunlight in 1899: 143.62
Mean sunlight in 1999: 126.12
```

```
<ipython-input-9-1da6f540402b>:20: FutureWarning: The default value of regex
will change from True to False in a future version. In addition, single
character regular expressions will *not* be treated as literal strings when
regex=True.
  weather_df.rain = weather_df['rain'].str.replace('*','')
<ipython-input-9-1da6f540402b>:21: FutureWarning: The default value of regex
will change from True to False in a future version. In addition, single
character regular expressions will *not* be treated as literal strings when
regex=True.
  weather_df.sun = weather_df['sun'].str.replace('*','')
<ipython-input-9-1da6f540402b>:22: FutureWarning: The default value of regex
will change from True to False in a future version. In addition, single
character regular expressions will *not* be treated as literal strings when
regex=True.
  weather_df.sun = weather_df['sun'].str.replace('$','')
/opt/conda/lib/python3.9/site-packages/pandas/util/_decorators.py:311:
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#returning-a-view-versus-a-copy
  return func(*args, **kwargs)
<ipython-input-9-1da6f540402b>:31: 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#returning-a-view-versus-a-copy
  sun_df['sun'] = sun_df.sun.astype('float')
<ipython-input-9-1da6f540402b>:36: 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#returning-a-view-versus-a-copy
  rain_df['rain'] = rain_df.rain.astype('float')
```

Your written answer for Q2.1 goes here

The mean rainfall in 1850s: 58.65. The mean rainfall in 1990s: 76.11. The mean sunlight in 1899: 143.62. The mean sunlight in 1999: 126.12

Calculations are using 31094 rows for rain and 28769 rows for sun.

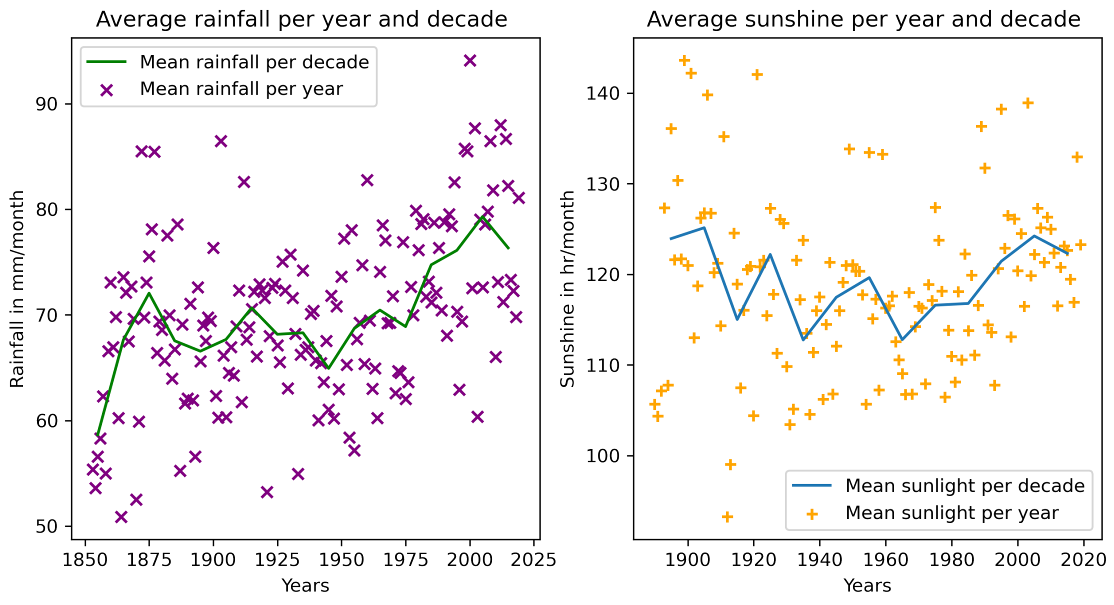
Question 2.2 (9 points)

Plot the mean rainfall and sunlight per year and per decade, in mm/month and in hours/month, respectively. Make two plots, one for rainfall and one for sunlight, with both plots having both yearly and per-decade averages.

```
[46]: #Rain
avg_rain_decades = []
rain_decades = []
avg_rain_years = []
rain_years = []

#Sun
avg_sun_decades = []
sun_decades = []
avg_sun_years = []
sun_years = []

#loop to caclulate the mean rainfall and sunlight per decade
for i in range (1850, 2020, 10):
    dec_rain_df = rain_df[(rain_df['yyyy']>= i) & (rain_df['yyyy'] < i+10)]
    dec_rain_mean = round(dec_rain_df.rain.mean(), 2)
    avg_rain_decades.append(dec_rain_mean)
    rain_decades.append(i+5)

    dec_sun_df = sun_df[(sun_df['yyyy']>= i) & (sun_df['yyyy'] < i+10)]
    dec_sun_mean = round(dec_sun_df.sun.mean(), 2)
    avg_sun_decades.append(dec_sun_mean)
    sun_decades.append(i+5)

#loop to caclulate the mean rainfall and sunlight per year
for i in range (1850, 2020):
    yr_rain_df = rain_df[(rain_df['yyyy'] == i)]
    yr_rain_mean = round(yr_rain_df.rain.mean(), 2)
    avg_rain_years.append(yr_rain_mean)
    rain_years.append(i)

    yr_sun_df = sun_df[(sun_df['yyyy'] == i)]
    yr_sun_mean = round(yr_sun_df.sun.mean(), 2)
    avg_sun_years.append(yr_sun_mean)
    sun_years.append(i)

fig, (ax1, ax2) = plt.subplots(1,2, figsize=(10, 5))

ax1.scatter(rain_years, avg_rain_years, color = 'purple', label = 'Mean␣
 ↪rainfall per year', marker = 'x')
ax1.plot(rain_decades, avg_rain_decades, color = 'green', label = 'Mean␣
 ↪rainfall per decade')
```

```
ax2.scatter(sun_years, avg_sun_years, color = 'orange', label = 'Mean sunlight␣
  ↪per year', marker = '+')
ax2.plot(sun_decades, avg_sun_decades, label = 'Mean sunlight per decade')

ax1.title.set_text('Average rainfall per year and decade ')
ax1.set_xlabel('Years')
ax1.set_ylabel('Rainfall in mm/month')
ax1.legend()

ax2.title.set_text('Average sunshine per year and decade ')
ax2.set_xlabel('Years')
ax2.set_ylabel('Sunshine in hr/month')
ax2.legend()
```

```
[46]: <matplotlib.legend.Legend at 0x7f93aec33790>
```



Question 2.3 (6 points)

Describe the trends you see. Comment on limitations of the data. Are there any outliers you can explain?

Your answer to Q2.3 goes here

The mean rainfall has increased steadily over the past 75 years. There doesn't appear to be any outliers in this graph.

The sunlight has made a slightly fall overall however it had dipped in the middle of this time period.

A limitation of the above graphs is that both the rainfall and sunshine are measured in different units, mm and hours. This makes both graphs incomparable.

Another limitation of the data is that each point plotted is a mean and that some of the accuracy is lost.

The sunshine data has a lot of variance in the years 1880 - 1930, this could be for a number of reasons possibly because the instruments used were very unreliable and never produced any consistent data.

Question 3 (10 points)

Now we want to compare the data to a different data set on the UK climate. It comes from https://www.metoffice.gov.uk/research/climate/maps-and-data/uk-and-regional-series. Load it from the file `UK.txt` in the folder `weather_uk-wide`. It gives the sunlight for each month, season and year between 1919 and 2021.

Questions 3.1 and 3.2 can be solved together, so make sure you read through both before starting.

Question 3.1 (2 points)

Compute the mean sunlight for the 1910s decade and the 1990s decade of the UK-wide data (one number per decade), again in hours/month. Use the annual totals to calculate the average. Two decimal places are sufficient.

Hint: `pd.read_fwf()` can be used to read `.txt` files.

[11]:
```python
# Your code for Q3.1 goes here
import statistics

uk_data = pd.read_fwf('weather_uk-wide/UK.txt', skiprows = 5)

b_sunlight_df = uk_data[(uk_data['year']>= 1910) & (uk_data['year'] < 1920)]
b_sunlight_mean = round(b_sunlight_df.ann.mean(), 2)

print("Mean sunfall in 1910s:", b_sunlight_mean)

a_sunlight_df = uk_data[(uk_data['year']>= 1990) & (uk_data['year'] < 2000)]
a_sunlight_mean = round(a_sunlight_df.ann.mean(), 2)

print("Mean sunfall in 1990s:", a_sunlight_mean)
```

```
Mean sunfall in 1910s: 1381.1
Mean sunfall in 1990s: 1368.21
```

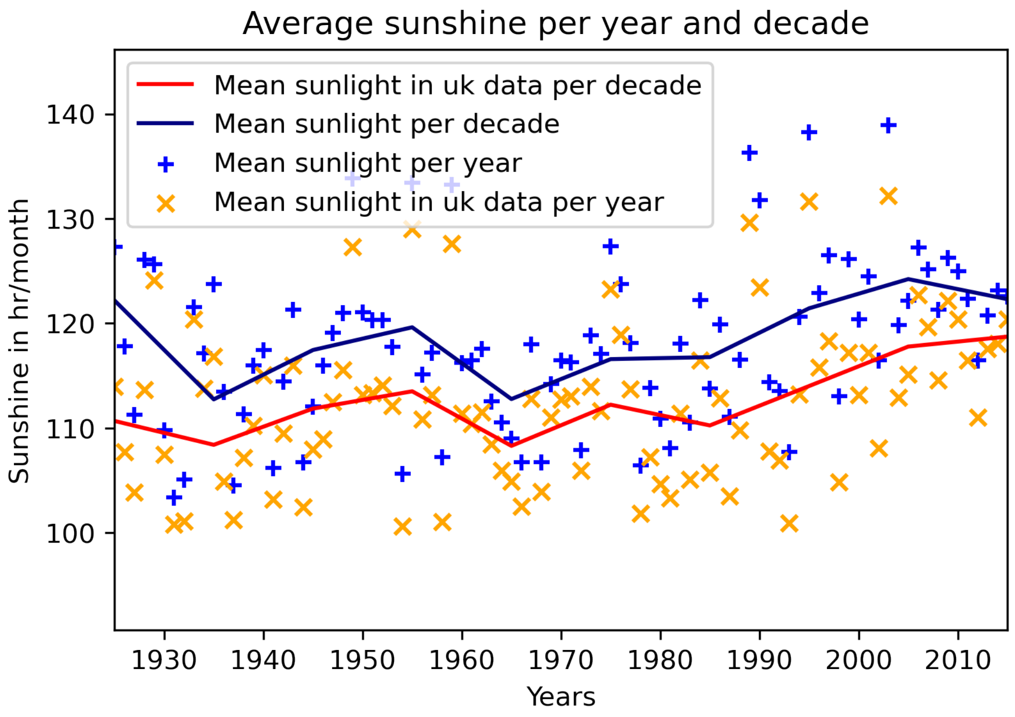
Your written answer for Q3.1 goes here

Question 3.2 (5 points)

Make a copy of the sunlight plot from the question 2.2 and add the UK-wide values.

[166]:
```python
uk_data_sun_decades=[]
uk_avg_sun_decades=[]

for i in range (1920, 2020, 10):
    uk_data_sun_decades.append(i+5)
    uk_dec_sun_df = uk_data[(uk_data['year']>= i) & (uk_data['year'] < i+10)]
    uk_dec_sun_mean = round(((uk_dec_sun_df.ann)/12).mean(), 2)
    uk_avg_sun_decades.append(uk_dec_sun_mean)

plt.scatter(sun_years, avg_sun_years, color = 'blue', label = 'Mean sunlight␣
 ↪per year', marker = '+')
plt.scatter(uk_data.year, (uk_data.ann)/12, color = 'orange', label = 'Mean␣
 ↪sunlight in uk data per year', marker = 'x')
plt.plot(uk_data_sun_decades, uk_avg_sun_decades, color = 'red', label = 'Mean␣
 ↪sunlight in uk data per decade')
plt.plot(sun_decades, avg_sun_decades, color='navy', label = 'Mean sunlight per␣
 ↪decade')

plt.title('Average sunshine per year and decade ')
plt.xlabel('Years')
plt.ylabel('Sunshine in hr/month')
plt.legend()
plt.xlim([1925, 2015])
```

[166]:
```
(1925.0, 2015.0)
```


Average sunshine per year and decade
Mean sunlight in uk data per decade
Mean sunlight per decade
Mean sunlight per year
Mean sunlight in uk data per year
Sunshine in hr/month
100
110
120
130
140
1930
1940
1950
1960
1970
1980
1990
2000
2010
Years

Question 3.3 (3 points)

Describe the trends you see. How do you explain the differences, given that both data sets come from the UK Met Office?

Overall, both the sets of data produce quite similar results. The patterns of the data are very similar as it increases and decreases over the same decade period. The main difference is that the data from the uk data set has produced lower values consistently, resulting in a lower plot. This is potentially the more accurate result as it has included more data.

Part C - Exploring European migration patterns

Questions 4 and 5 focus on migration statistics in Europe between the years 1990 and 2019 from Eurostat. The Eurostat dataset we use contains tables for immigration and emigration across different countries for these three decades (https://ec.europa.eu/eurostat/databrowser/view/MIGR_IMM8___custom_1301560/default/table?lang=en and https://ec.europa.eu/eurostat/databrowser/view/MIGR_EMI2___custom_1301550/default/table?lang=en). In Question 5 we also use a dataset that provides more information about individual countries.

Question 4 (24 points)

In this question we will focus on migration trends throughout Europe. We have already merged the data for immigration and emigration for you, which is presented in the `migration_data/EUROSTAT_migrants.csv` file. Furthermore, variables not in use in this exercise were removed and columns were renamed for your convenience. The `EUROSTAT_migrants.csv` file includes the age group (at the time of migration), sex of the migrant, the country code, the year, and the number of immigrants (`count_in`) and emigrants (`count_out`).

Question 4.1 (3 points)

State the total number of male and female immigrants with known age present in the dataset. “Known age” here means belonging to the age groups `Y_LT1`, `Y_GE100`, and `Y_{num}`, where `num` is an integer such that `0<num<100`.

```
[12]: # Reading in the csv file
EUROSTAT_migrants_loc = os.path.join(os.getcwd(), 'migration_data',␣
    ↪'EUROSTAT_migrants.csv')
EUROSTAT_migrants = pd.read_csv(EUROSTAT_migrants_loc)

discard = ['UNK', 'TOTAL']
known_age_migrant = EUROSTAT_migrants[~EUROSTAT_migrants.age_group.str.
    ↪contains('|'.join(discard),  na=False)]

f_known_age_migrant = known_age_migrant[known_age_migrant.sex == 'F']
m_known_age_migrant = known_age_migrant[known_age_migrant.sex == 'M']

print("The total number of female immigrants with known age is",␣
    ↪f_known_age_migrant.count_in.sum())
print("the total number of male immigrants with known age is",␣
    ↪m_known_age_migrant.count_in.sum())
```

```
The total number of female immigrants with known age is 24947967
the total number of male immigrants with known age is 29901443
```

Your written answer for Q4.1 goes here

24947967 female immigrants

29901443 male immigrants

Question 4.2 (5 points)

State which known age group contains the highest number of immigrants, and the respective number of immigrants. Do this for each sex.

[13]:
```
f_known_age_migrant.age_group = f_known_age_migrant.age_group.str.
 ↪replace('Y','')
m_known_age_migrant.age_group = m_known_age_migrant.age_group.str.
 ↪replace('Y','')

LT1_f_df = f_known_age_migrant[f_known_age_migrant.age_group == '_LT1']
LT1_f = LT1_f_df.count_in.sum()

LT1_m_df = m_known_age_migrant[m_known_age_migrant.age_group == '_LT1']
LT1_m = LT1_m_df.count_in.sum()

GE100_f_df = f_known_age_migrant[f_known_age_migrant.age_group == '_GE100']
GE100_f = GE100_f_df.count_in.sum()

GE100_m_df = m_known_age_migrant[m_known_age_migrant.age_group == '_GE100']
GE100_m = GE100_m_df.count_in.sum()

discard = ['_LT1', '_GE100']
f_clean = f_known_age_migrant[~f_known_age_migrant.age_group.str.contains('|'.
 ↪join(discard),  na=False)]
m_clean = m_known_age_migrant[~m_known_age_migrant.age_group.str.contains('|'.
 ↪join(discard),  na=False)]

highest_m = 0
highest_m_age = 0
highest_f = 0
highest_f_age = 0
f_sums = []
m_sums = []
age = []

for x in range (1, 100):
    f_age_subset = f_clean[f_clean.age_group.astype('float') == x]
    f_sum = f_age_subset.count_in.sum()
    f_sums.append(f_sum)
    age.append(x)
    if f_sum > highest_f:
        highest_f = f_sum
        highest_f_age = x

    m_age_subset = m_clean[m_clean.age_group.astype('float') == x]
    m_sum = m_age_subset.count_in.sum()
    m_sums.append(m_sum)
    if m_sum > highest_m:
        highest_m = m_sum
        highest_m_age = x

if highest_f <  GE100_f:
    highest_f_age = "100+"
elif  highest_f <  LT1_f:
    highest_f_age = "1-"

if highest_m <  GE100_m:
    highest_m_age = "100+"
elif  highest_m <  LT1_m:
    highest_m_age = "1-"


print("The female highest number of known age imigrants are aged",␣
 ↪highest_f_age, "with", highest_f, "values")
print("The male highest number of known age imigrants are aged", highest_m_age,␣
 ↪"with", highest_m, "values")
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/generic.py:5507:
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#returning-a-view-versus-a-copy
  self[name] = value
```

```
The female highest number of known age imigrants are aged 24 with 926900 values
The male highest number of known age imigrants are aged 26 with 1063230 values
```

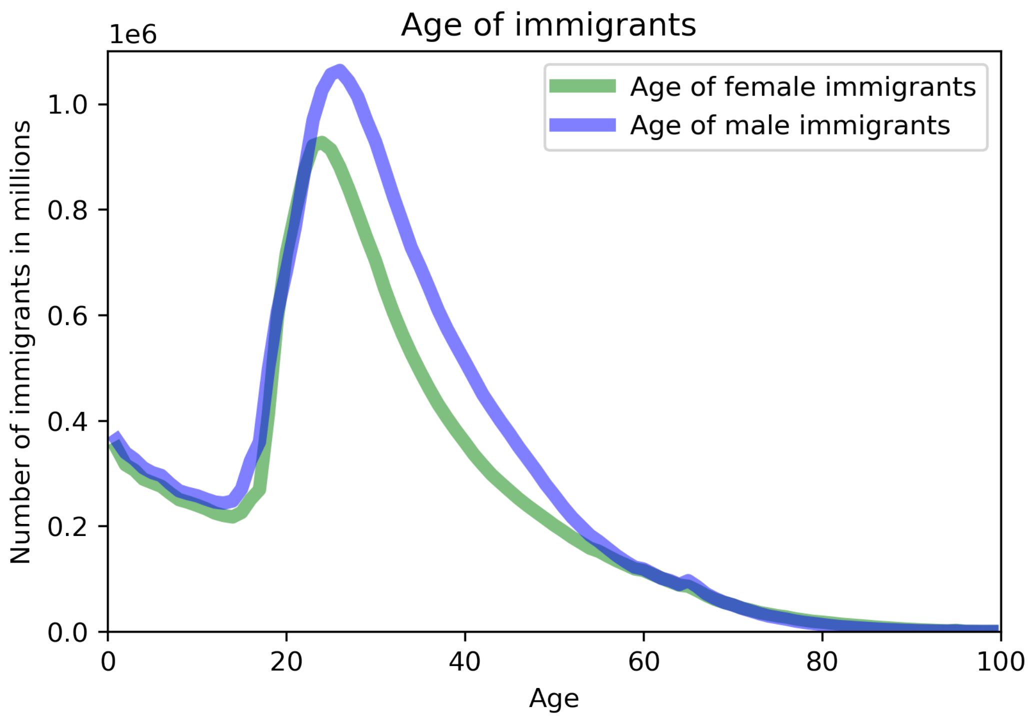
The female highest number of known age imigrants are aged 24 with 926900 values. The male highest number of known age imigrants are aged 26 with 1063230 values.

Question 4.3 (10 points)

Plot the total number of male and female immigrants across different known-age groups.

[22]:
```
#plotting the graph
plt.plot(age, f_sums, color = 'green', label = 'Age of female immigrants',␣
 ↪alpha=0.5, lw = 5)
plt.plot(age, m_sums, color = 'blue', label = 'Age of male immigrants', alpha=0.
 ↪5, lw = 5)

plt.title('Age of immigrants ')
plt.xlabel('Age')
plt.ylabel('Number of immigrants in millions')
plt.legend()
plt.xlim([0, 100])
plt.ylim([0,1100000])
```

[22]:
```
(0.0, 1100000.0)
```



Question 4.4 (6 points)

Interpret the plot, identify interesting patterns within it, and comment on why these patterns might be present.

On first glance, both these graphs are fairly similar and it is clear that they both follow the same trends. However the number of male immigrants from about aged 24 until aged 55 is significantly higher that the number of female immigrants. Looking at the graph from 0 to 18, the number of child immigrants dips by a large percentage. This could potentially be because when families are choosing to migrate, young children have no choice not to move, however as children are older they may have more freedom to choose whether they would like to move or not. Also at the age where it dips, the children are at school so families may choose not to migrate while they at this age. However from aged 18 until 25, for female, and 30 for male, there is a huge rise in the number of migrants. This is most likely because a large percentage of adults will move country to work, or to go to university. This begins to fall more gradually from this age until 65. This is most likely because many adults will settle at this age into a country and will not relocate as often. There is also a small rise mostly for male migrants at age 65, this is around the age of retirement so it is tangible that this is because many people have chosen to move countries at this age. Then the graphs both slowly fall until they are negligible.

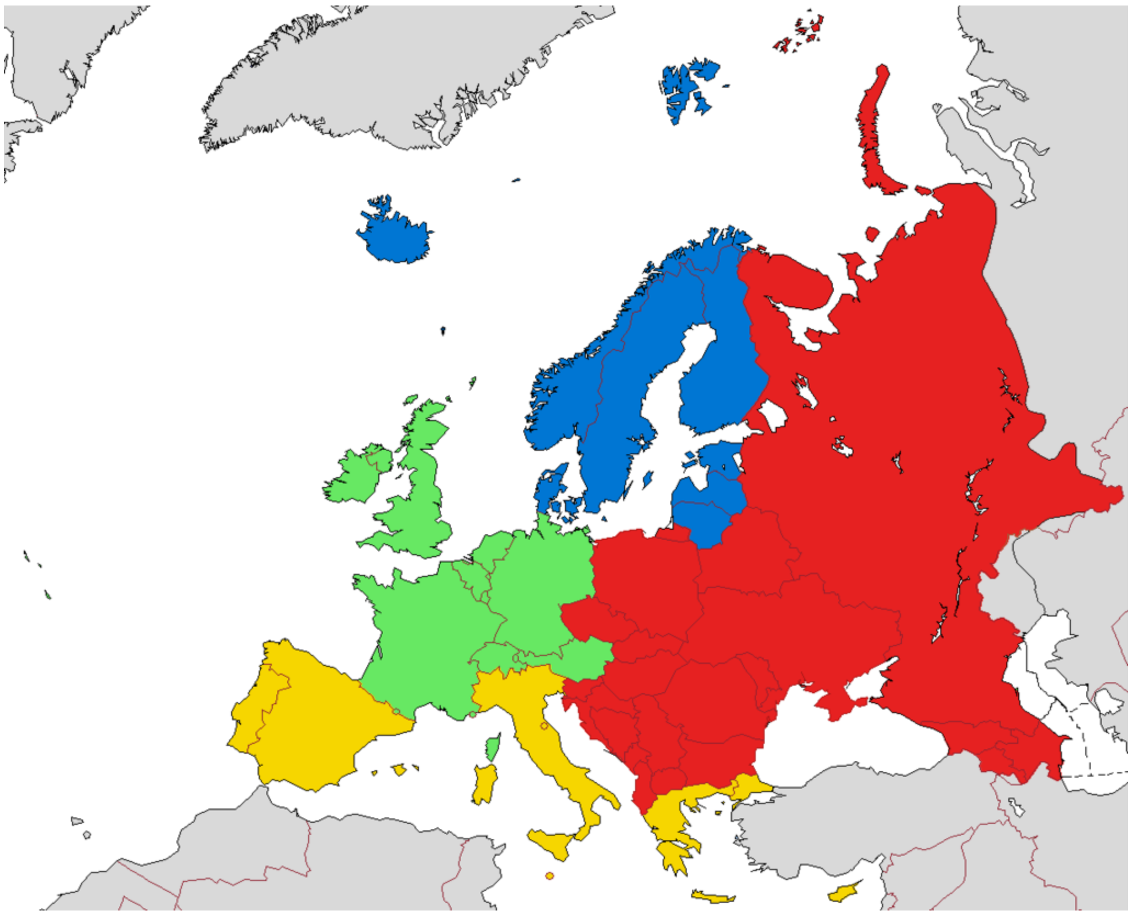
Question 5 (26 points)

In this exercise we wish to compare how migration has changed in different European regions.

We provide further information about individual countries in the file `migration_data/Countries.csv`. This includes: the country's name; the country code; its population in 2011 stated in millions of inhabitants; and what larger region it belongs to according to the multidisciplinary thesaurus (controlled vocabulary) EuroVoc. The idea is that the four regions (North, West, South, East) have different population sizes, which directly affects the number of migrants associated with each region.

Firstly, we want to find the net migration of each region for each year, and scale it appropriately. By "appropriately" we mean that the scaling factor should be the sum of the populations of all contributing countries from that region for that year - e.g., Czechia does not provide any data for year 1991, so its population will not be considered in the scaling factor for Eastern Europe for that year. On the other hand, Slovakia and Croatia are countries classified in the Eastern Europe region with data available for 1991, hence their individual populations will contribute towards the sum that represents the region's scaling factor for that year.

Once we have all the net migration values scaled for each region for each of the 30 years within our dataset, we want a plot of the distributions of the scaled net migration for the four regions over the three decades - one distribution per region per decade.



Question 5.1 (3 points)

Load `EUROSTAT_migration.csv` afresh. State the sum of net migration (migrants coming in minus migrants going out of each country) across Europe over the past three decades according to the data. Use the data with age group TOTAL.

[26]:
```
# reading in the csv file
eurostat_migrants_loc = os.path.join(os.getcwd(), 'migration_data',␣
  ↪'EUROSTAT_migrants.csv')
eurostat_migrants = pd.read_csv(eurostat_migrants_loc)
eurostat_migrants = eurostat_migrants[eurostat_migrants.age_group == "TOTAL"]

lst = eurostat_migrants.groupby('country')['country'].apply(pd.Series.tolist).
  ↪tolist()
countries = []

for y in range (0, len(lst)):
    countries.append(lst[y][0])

grouped = eurostat_migrants.groupby('country')
count_in = grouped['count_in'].sum()
count_out = grouped['count_out'].sum()

#working out the net migration
net_migration = []
for x in range(0, len(count_in)):
    net_migration.append((count_in[x]) - (count_out[x]))

for z in range (0, len(net_migration)):
    print(countries[z], net_migration[z])
```

```
AT 822136.0
BE 985383.0
BG -32372.0
CH 1145137.0
CY 47905.0
CZ 346952.0
DE 6543578.0
DK 382498.0
EE -8509.0
EL -143689.0
ES 4392568.0
FI 298324.0
FR 974370.0
HR 205697.0
HU 456207.0
IE 516636.0
IS 43217.0
```

```
IT 5895050.0
LI 2072.0
LT -630392.0
LU 195657.0
LV -452400.0
MK 8970.0
MT 98318.0
NL 1430717.0
NO 660928.0
PL -689798.0
PT 404631.0
RO -705681.0
SE 1326714.0
SI 118651.0
SK 74902.0
UK 4387330.0
```

The total sum of net migration is 29101707.

Question 5.2 (3 points)

Merge the migration and countries datasets. State how many countries represent each region in the merged dataset.

[27]:
```
#readining in csv
countries_loc = os.path.join(os.getcwd(), 'migration_data', 'Countries.csv')
countries = pd.read_csv(countries_loc)

#changing the name of the data frame title so that the two data sets can be␣
  ↪merged
countries.rename(columns={'country_code': 'country'}, inplace=True)
migration_countries = (pd.merge(eurostat_migrants, countries))

print(migration_countries)

#grouping each region by country
grouped_n_region = migration_countries.groupby('country')

north_region = migration_countries[migration_countries['EuroVoc Region'] ==␣
  ↪'North']
south_region = migration_countries[migration_countries['EuroVoc Region'] ==␣
  ↪'South']
east_region = migration_countries[migration_countries['EuroVoc Region'] ==␣
  ↪'East']
west_region = migration_countries[migration_countries['EuroVoc Region'] ==␣
  ↪'West']

grouped_n_region = len(north_region.groupby(['country'])['country'].count())
grouped_s_region = len(south_region.groupby(['country'])['country'].count())
grouped_w_region = len(west_region.groupby(['country'])['country'].count())
grouped_e_region = len(east_region.groupby(['country'])['country'].count())

print("north:", grouped_n_region, "south:", grouped_s_region, "west:",␣
  ↪grouped_w_region, "east:", grouped_e_region)
```

```
    age_group sex country  year  count_in  count_out       Full Name  \
0       TOTAL   F      AT  1996     32883    27782.0         Austria
1       TOTAL   F      AT  1997     32392    29220.0         Austria
2       TOTAL   F      AT  1998     33793    27736.0         Austria
3       TOTAL   F      AT  1999     41346    29750.0         Austria
4       TOTAL   F      AT  2000     38157    28755.0         Austria
…         … ..   …     …     …        …          …             …
1457    TOTAL   M      UK  2015    325158   163722.0  United Kingdom
1458    TOTAL   M      UK  2016    313466   190007.0  United Kingdom
1459    TOTAL   M      UK  2017    346661   199427.0  United Kingdom
1460    TOTAL   M      UK  2018    299747   190477.0  United Kingdom
1461    TOTAL   M      UK  2019    336038   201223.0  United Kingdom
```

```
     EuroVoc Region  Population (2011)
0              West              8.375
1              West              8.375
2              West              8.375
3              West              8.375
4              West              8.375
…              …                 …
1457           West             63.020
1458           West             63.020
1459           West             63.020
1460           West             63.020
1461           West             63.020

[1462 rows x 9 columns]
north: 8 south: 6 west: 9 east: 9
```

North: 8 countries, South: 6 countries, West: 9 countries, East: 9 countries,

Question 5.3 (5 points)

Compute the scaling factors for each region for each year. The scaling factor is the sum of the populations of the countries within the region that contributed data for that year. Assume the population of each country is constant across the years and is equal to the 2011 population data provided in `Countries.csv`. State: - the maximum and minimum scaling factor of net migration in Eastern Europe - the years in which the maximum and the minimum occur.

```
[28]: #grouping the each region by the country to workout the sum of each countries␣
       ↪population
      scaling_n_region = (north_region.groupby(['country'])['Population (2011)'].
       ↪mean()).sum()
      scaling_s_region = (south_region.groupby(['country'])['Population (2011)'].
       ↪mean()).sum()
      scaling_w_region = (west_region.groupby(['country'])['Population (2011)'].
       ↪mean()).sum()
      scaling_e_region = (east_region.groupby(['country'])['Population (2011)'].
       ↪mean()).sum()

      print("The north scaling factor:", scaling_n_region)
      print("The south scaling factor:", scaling_s_region)
      print("The west scaling factor:", scaling_w_region)
      print("The east scaling factor:", scaling_e_region)

      #print(east_region)
      max_east=0
      max_east_year=0
      min_east=0
      min_east_year=0

      grouped_e_y = east_region.groupby('year')
      count_in_e = (grouped_e_y['count_in'].sum()).astype(float)
      count_out_e = grouped_e_y['count_out'].sum()

      net_migration_e = []
      for x in range(1990, (1990 + len(count_in_e))):
          net_migration_e.append(((count_in_e[x]) - (count_out_e[x])) /␣
       ↪scaling_e_region)

      #creating a years
      years = []
      for x in range (1990, 2020):
          years.append(x)

      for x in range (0, len(net_migration_e)):
          if max_east < net_migration_e[x]:
              max_east = net_migration_e[x]
```

```
        max_east_year = 1990+x

    if min_east > net_migration_e[x]:
        min_east = net_migration_e[x]
        min_east_year = 1990+x

print("The maximum Easturn Europe scaling factor of net migration:", max_east,␣
 ↪"in", max_east_year)
print("The minimum Easturn Europe scaling factor of net migration:", min_east,␣
 ↪"in", min_east_year)
```

```
The north scaling factor: 32.043
The south scaling factor: 128.975751
The west scaling factor: 257.092149
The east scaling factor: 99.894
The maximum Easturn Europe scaling factor of net migration: 1087.0222435781927
in 2007
The minimum Easturn Europe scaling factor of net migration: -1716.3092878451157
in 2011
```

The north scaling factor: 32.043, The south scaling factor: 128.975751, The west scaling factor: 257.092149, The east scaling factor: 99.894, The maximum Easturn Europe scaling factor: 6543578.0, The minimum Easturn Europe scaling factor: -705681.0,

Question 5.4 (9 points)

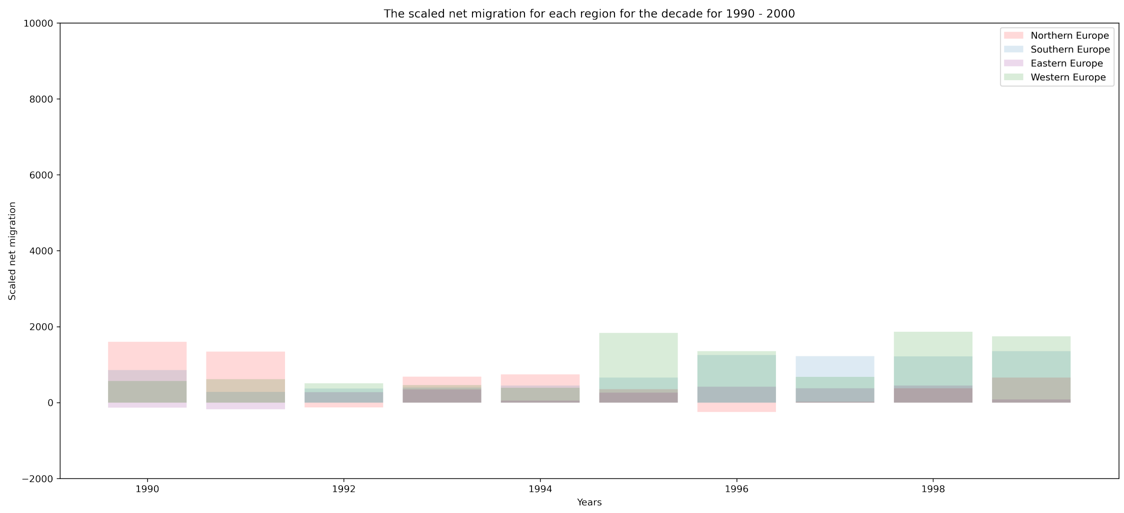
Compute the scaled net migration for each year for each region. Plot the distribution of scaled net migration for each region for each of the three decades (1990s, 2000s, 2010s).

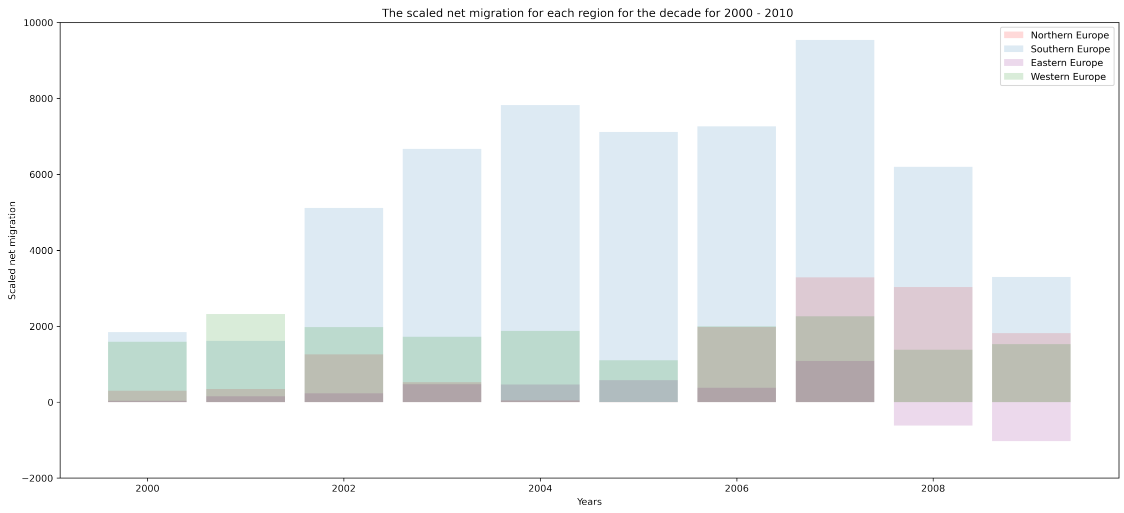
[49]:
```
#north region in subsets of decades, grouped by year
grouped_n_y = north_region.groupby('year')
count_in_n = (grouped_n_y['count_in'].sum()).astype(float)
count_out_n = grouped_n_y['count_out'].sum()

net_migration_n = []
for x in range(1990, (1990 + len(count_in_n))):
    net_migration_n.append(((count_in_n[x]) - (count_out_n[x])) /␣
 ↪scaling_n_region)

north_1990 = net_migration_n[0:10]
north_2000 = net_migration_n[10:20]
north_2010 = net_migration_n[20:30]

#south region in subsets of decades
grouped_s_y = south_region.groupby('year')
count_in_s = (grouped_s_y['count_in'].sum()).astype(float)
count_out_s = grouped_s_y['count_out'].sum()

net_migration_s = []
for x in range(1990, (1990 + len(count_in_s))):
    net_migration_s.append(((count_in_s[x]) - (count_out_s[x])) /␣
 ↪scaling_s_region)

south_1990 = net_migration_s[0:10]
south_2000 = net_migration_s[10:20]
south_2010 = net_migration_s[20:30]

#east region in subsets of decades
east_1990 = net_migration_e[0:10]
east_2000 = net_migration_e[10:20]
east_2010 = net_migration_e[20:30]

#west region in subsets of decades
grouped_w_y = west_region.groupby('year')
count_in_w = (grouped_w_y['count_in'].sum()).astype(float)
count_out_w = grouped_w_y['count_out'].sum()

net_migration_w = []
for x in range(1990, (1990 + len(count_in_w))):
    net_migration_w.append(((count_in_w[x]) - (count_out_w[x])) /␣
 ↪scaling_w_region)
```

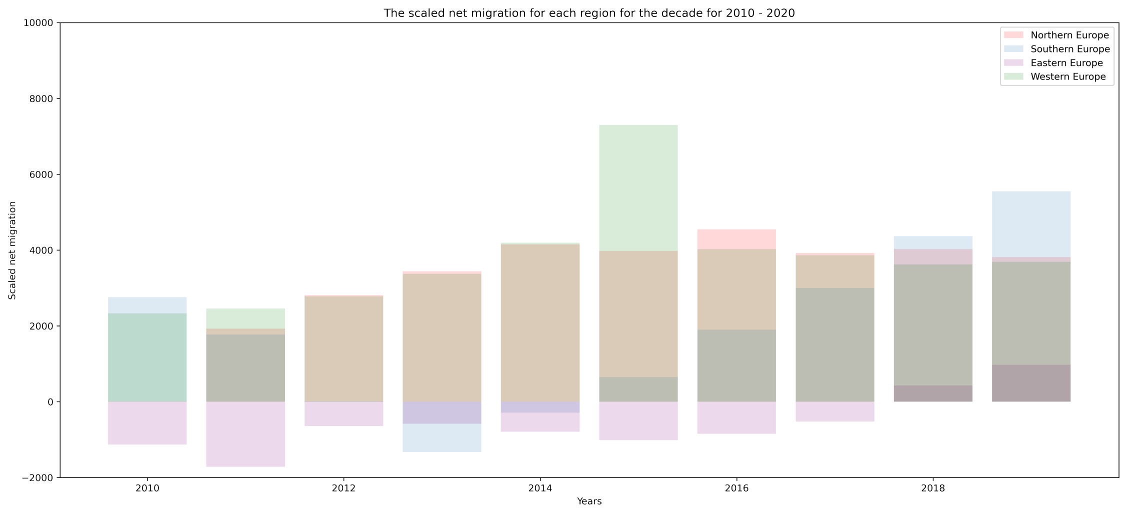
```
west_1990 = net_migration_w[0:10]
west_2000 = net_migration_w[10:20]
west_2010 = net_migration_w[20:30]


#print(net_migration_n)


#plotting the graphs for each decade, colour co-ordinated by region.
fig, ((ax1), (ax2), (ax3)) = plt.subplots(3,1, figsize=(20, 30))

ax1.bar(years[0:10], north_1990, color = 'red', alpha = 0.15, label = 'Northern␣
 ↪Europe')
ax1.bar(years[0:10], south_1990, alpha = 0.15, label = 'Southern Europe')
ax1.bar(years[0:10], east_1990, color = 'purple', alpha = 0.15, label =␣
 ↪'Eastern Europe')
ax1.bar(years[0:10], west_1990, color = 'green', alpha = 0.15, label = 'Western␣
 ↪Europe')

ax2.bar(years[10:20], north_2000, color = 'red', alpha = 0.15, label =␣
 ↪'Northern Europe')
ax2.bar(years[10:20], south_2000, alpha = 0.15, label = 'Southern Europe')
ax2.bar(years[10:20], east_2000, color = 'purple', alpha = 0.15, label =␣
 ↪'Eastern Europe')
ax2.bar(years[10:20], west_2000, color = 'green', alpha = 0.15, label =␣
 ↪'Western Europe')

ax3.bar(years[20:30], north_2010, color = 'red', alpha = 0.15, label =␣
 ↪'Northern Europe')
ax3.bar(years[20:30], south_2010, alpha = 0.15, label = 'Southern Europe')
ax3.bar(years[20:30], east_2010, color = 'purple', alpha = 0.15, label =␣
 ↪'Eastern Europe')
ax3.bar(years[20:30], west_2010, color = 'green', alpha = 0.15, label =␣
 ↪'Western Europe')

ax1.set_title("The scaled net migration for each region for the decade for 1990␣
 ↪- 2000")
ax2.set_title("The scaled net migration for each region for the decade for 2000␣
 ↪- 2010 ")
ax3.set_title("The scaled net migration for each region for the decade for 2010␣
 ↪- 2020")

ax1.set_ylim([-2000, 10000])
ax2.set_ylim([-2000, 10000])
ax3.set_ylim([-2000, 10000])
```

```
ax1.set_xlabel('Years')
ax1.set_ylabel('Scaled net migration in millions')
ax1.legend()

ax2.set_xlabel('Years')
ax2.set_ylabel('Scaled net migration in millions')
ax2.legend()

ax3.set_xlabel('Years')
ax3.set_ylabel('Scaled net migration in millions')
ax3.legend()
```

```
[49]: <matplotlib.legend.Legend at 0x7f93b2236520>
```


The scaled net migration for each region for the decade for 1990 - 2000
Northern Europe
Southern Europe
Eastern Europe
Western Europe
Scaled net migration
Years


The scaled net migration for each region for the decade for 2000 - 2010
Northern Europe
Southern Europe
Eastern Europe
Western Europe
Scaled net migration
Years


The scaled net migration for each region for the decade for 2010 - 2020
Northern Europe
Southern Europe
Eastern Europe
Western Europe
Scaled net migration
Years

Question 5.5 (6 points)

Analyse and interpret the plot. Are there any interesting patterns? Can you link them to some economic or geopolitical events?

Looking at the first decade plotted from 1990 - 2000, we can see that at the beginning of the decade Northern Europe had the biggest increase in net migration, however in the 1996 there was an overall decrease in this region. Western Europe peaked for scaled net migration in 1995. All the other regions have had a fluctuated lightly.

For the next decade, 2000 - 2010, Southern Europe has increased but has peaked significantly in 2007 this could be as a result of the 2008 great recession. Also there is a general decrease of scaled net migration in 2008 and 2009. This could be as a result of migrants moving from western to eastern europe as a result of the great recession.

Finally looking at the last decade 2010-2020, Western Europe has continued to have a negative scaled net migration. This could be due to the refugee crisis, in 2011, which caused many Syrian refugees to leave their country which caused them to relocate in Europe. There has been a gradual increase in the scaled net migration in Northern Europe. In 2015 there was a spike in scaled net migration in wester Europe also.