

Introduction

✓ Data science is a powerful tool used by researchers in various fields to test scientific hypotheses. It is a multi-step process which involves pre-processing large data samples, analysing the data samples with various regression tools and then visualizing the results. Here we will go through an example case of using data science to see the effect of COVID-19 pandemic on the quality of air we breathe.



Pictures taken over Milan, Italy, before (left) and after (after) COVID-19 pandemic

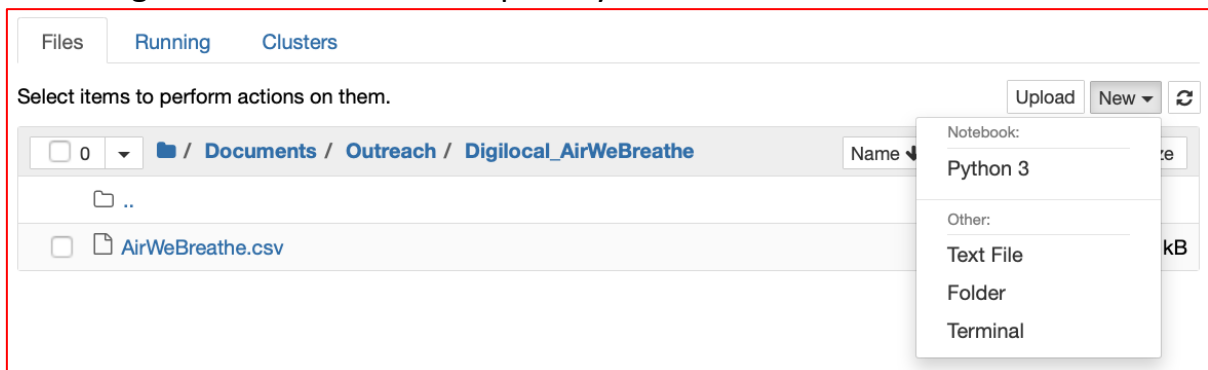
✓ The COVID-19 pandemic started affecting the world in early 2020. Various countries started implementing lockdown measures to decelerate the spread of this pandemic. The lockdown measures involved restrictions on unessential travels. Travel vehicles emit various pollutant gases such as nitrogen dioxide (NO_2) which directly affect the quality of air we breathe (forming harmful particles) and our health (inflammation of airways). Here we will use data science capability of Python to see the effect of the travel restrictions on NO_2 levels in Bristol and London, two largest cities in southern England.

✓ The Department of Environment Food and Rural Affairs (DEFRA) website (<https://uk-air.defra.gov.uk/>) provides free access to measurement data of various pollutants at various sites within the UK. A csv file containing NO_2

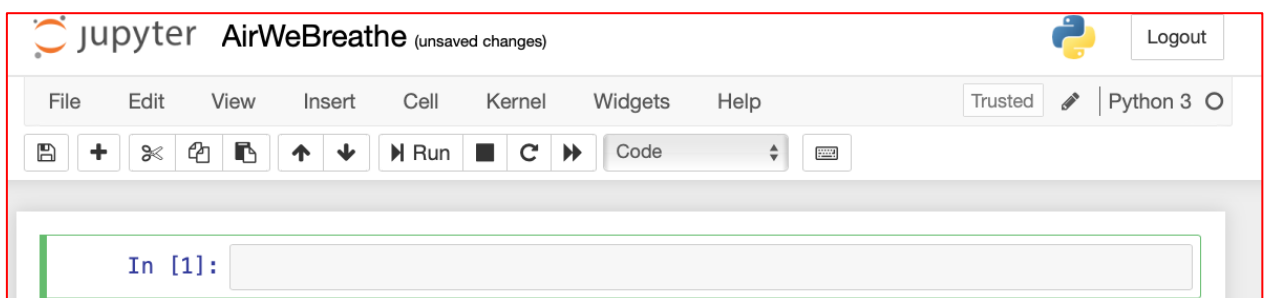
measurements from the sites at Bristol Temple Way and London Marylebone Road is provided in the resources folder for this project. Detail instructions on accessing data from the DEFRA website are provided in the Primary Science Teaching Trust website (<https://pstt.org.uk/resources/curriculum-materials/post-16-citizen-science-air-pollution>).

Step 1: Getting Started with Jupyter

✓ Jupyter is an interactive development environment for Python and we will use it to write and execute our Python program for data analysis. Before using Jupyter, create a folder named AirWeBreathe inside your named folder to save all the files for this project. Jupyter will open in an internet browser and you should browse to the AirWeBreathe folder we created earlier. After you are inside your AirWeBreathe folder which should be empty, upload the AirWeBreathe.csv file from the python resource folder using the upload button on the right-hand corner. After upload you should see the csv file.



✓ Create a new Python 3 notebook using the dropdown option under New. This will open a new tab in the browser and a Jupyter notebook titled Untitled will be created. Go back to the Jupyter tab and select the Untitled notebook and rename (under Files option) it to AirWeBreathe.ipynb



Step 2: Importing Modules and Data

✓ Now we will import numpy and pandas which are very popular modules for data analysis in Python. NumPy is a standard module in Python whereas you may need to install the pandas module manually before importing. Shown below are links to NumPy and pandas manuals and installation guide:

<https://numpy.org/>

<https://pandas.pydata.org/pandas-docs/version/0.23.3/install.html>

The empty cell (highlighted green when selected) takes the python commands and once you have typed the import commands execute it by using shift + return.

```
In [1]: import numpy as np
import pandas as pd
```

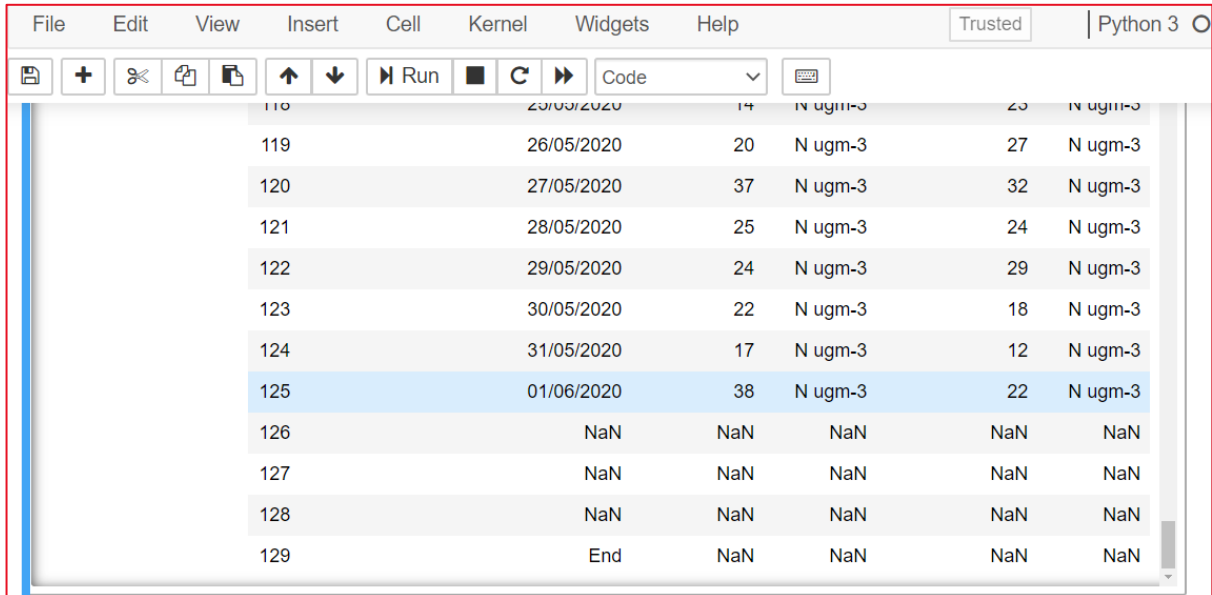
✓ The pandas module has read_csv function which we will use to import the NO₂ measurement data from the csv files into Python as a dataframe, df. We will also display the contents of the measurement file using display command. As the file has quite a lot of rows we need to change the setting of the display command in the second line of code to 1000.

```
In [24]: df = pd.read_csv('AirWeBreathe.csv')
pd.set_option('display.max_rows', 1000)
display(df)
```

	Daily Mean data\t supplied by UK-air on 28/06/2020	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	All Data GMT hour ending	NaN	NaN	NaN	NaN
1	Status: V=Verified P=Provisionally Verified N=N...	NaN	NaN	NaN	NaN
2	NaN	Bristol Temple Way	NaN	London Marylebone Road	NaN
3	Date	Nitrogen dioxide	Status	Nitrogen dioxide	Status
4	01/02/2020	16	V ugm-3	71	N ugm-3

✓ Now we can see the structure of our dataset collected from DEFRA. We can see the first column contains index number for all the dataset starting from 0. The first four rows only contain description of the data. After that we can see the columns are divided into Date, NO₂ measurement from Bristol Temple Way, Measurement Units, NO₂ measurement from London

Marylebone Road and Measurement Units. There is also a scroll bar on the right which we can use to explore all the rows of the dataset. Shown below are the last rows of the dataset.



Index	Date	Measurement 1	Measurement 2	Measurement 3	Measurement 4
118	26/05/2020	14	N ugm-3	23	N ugm-3
119	26/05/2020	20	N ugm-3	27	N ugm-3
120	27/05/2020	37	N ugm-3	32	N ugm-3
121	28/05/2020	25	N ugm-3	24	N ugm-3
122	29/05/2020	24	N ugm-3	29	N ugm-3
123	30/05/2020	22	N ugm-3	18	N ugm-3
124	31/05/2020	17	N ugm-3	12	N ugm-3
125	01/06/2020	38	N ugm-3	22	N ugm-3
126	NaN	NaN	NaN	NaN	NaN
127	NaN	NaN	NaN	NaN	NaN
128	NaN	NaN	NaN	NaN	NaN
129	End	NaN	NaN	NaN	NaN

✓ We can see that the first four rows of the file do not actually contain measurement data. As part of data pre-processing, a new dataframe, `sf`, is created in the first three lines of code below. The `dropna()` function drops any rows which have one or more empty values. This is critical for large datasets which may have random empty spaces. The `i` and `f` values inside the square bracket provide the range of index to be copied into the new dataframe. The fourth line re-indexes the dataframe so that it starts from 0 again.

```
In [22]: i=4 #initial index of the measurement data
          f=len(df) #final index of the measurement data
          sf=df[i:f].dropna() #dropping any rows with NaN value
          sf.reset_index(drop=True, inplace=True) #reindexing dataframe
          display(sf)
```

Challenge Time!
Verify that the `sf` dataframe is indexed properly using the `display` command

Step 3: Data Slicing

✓ Now we will slice the imported dataframe into separate arrays for dates and measurements from London. Array in Python is a collection of data of the same type. First the measurement data from London is sliced and saved to an array named 'London'. The `iloc` attribute of dataframe is used to select columns with index 1 and 3 for Bristol and London, respectively. The resulting dataframe series is then converted into NumPy 'float type' array using `to_numpy` command. This conversion will enable us to use various NumPy functions for further analysis of data later. We can see the values stored in the 'London' array using the print command

```
In [58]: London = sf.iloc[:,3].to_numpy('float')
          print(London)

[71. 68. 65. 44. 80. 84. 72. 65. 35. 69. 66. 74. 68. 77. 40. 55. 70. 74.
 76. 64. 74. 57. 40. 70. 86. 59. 54. 74. 61. 64. 61. 87. 90. 37. 62. 77.
 59. 79. 63. 75. 76. 69. 61. 53. 68. 66. 60. 29. 21. 34. 31. 65. 49. 61.
 32. 20.  9.  7. 15. 24. 33. 35. 42. 32. 23. 36. 42. 48. 44. 39. 37. 21.
  4. 24. 46. 44. 21. 17. 11. 14. 15. 18. 30. 27. 21. 25. 31. 19. 27. 27.
 27. 20. 24. 22. 21. 28. 43. 39. 37.  6.  5. 17.  9. 15. 23. 21. 23. 33.
 29. 38. 37. 26. 15. 13. 23. 27. 32. 24. 29. 18. 12. 22.]
```

✓ A similar procedure is applied to extract dates for the measurement. Unlike the measurement values, dates are provided in a different format. We will import the `datetime` module to change the dates into a format readable in python. The third line in the code below uses list comprehension in which every value stored in the dates series is operated by the datetime commands.

```
In [15]: dates = sf.iloc[:,0]
          import datetime as dt
          fmt_dates = [dt.datetime.strptime(d,'%d/%m/%Y').date() for d in dates]
          print(fmt_dates)
```

Shown below is the output of the print command

```
[datetime.date(2020, 2, 1), datetime.date(2020, 2, 2), datetime.date(2020,
2, 3), datetime.date(2020, 2, 4), datetime.date(2020, 2, 5), datetime.date
(2020, 2, 6), datetime.date(2020, 2, 7), datetime.date(2020, 2, 8), datetim
e.date(2020, 2, 9), datetime.date(2020, 2, 10), datetime.date(2020, 2, 11),
datetime.date(2020, 2, 12), datetime.date(2020, 2, 13), datetime.date(2020,
```

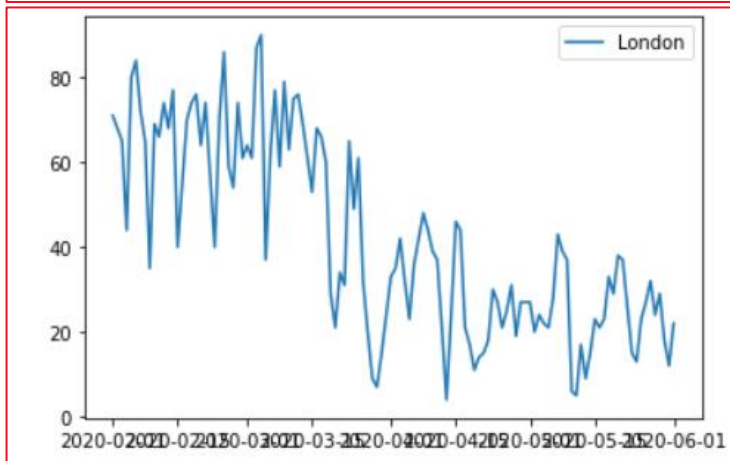

Challenge Time!

Slice the Bristol measurements from sf dataframe and save it to an array named 'Bristol'

Step 4: Data Visualisation

✓ We will now plot these arrays to visualise the measurements from London. We will import matplotlib module for this as shown in the first line below. A figure object, fig, is created and subplot, ax, is added to it. A plot labelled 'Bristol' is created in the ax subplot to visualise the Bristol NO₂ measurements. The plot and legend are displayed using the legend and show attributes of the matplotlib module

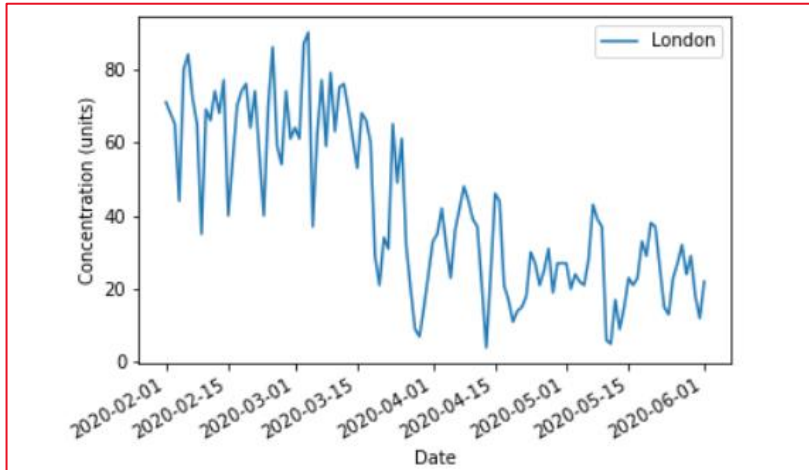
```
In [57]: import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(fmt_dates, London, label='London')
plt.legend()
plt.show()
```



✓ Further plot commands are added to display labels for the x (horizontal, xlabel) and y (vertical, ylabel) axis. The dates in the x axis is formatted for better visualisation using commands shown in line 7.

```
In [56]: import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(fmt_dates, Bristol, label='Bristol')
plt.xlabel('Date')
plt.ylabel('Concentration (units)')
plt.gcf().autofmt_xdate()
plt.legend()
plt.show()
```

The output plot is shown below which is much easier to read now!

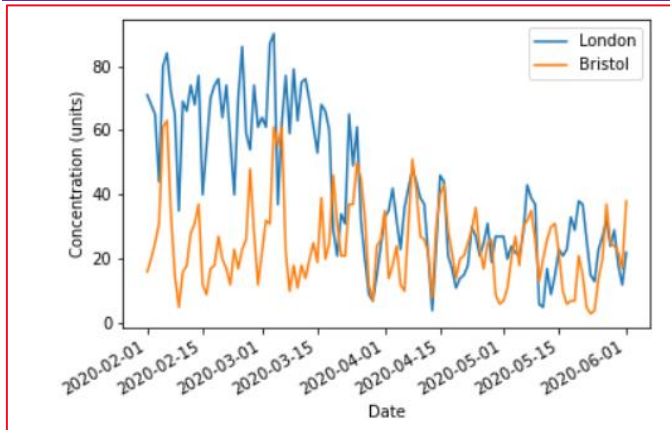


Science Alert!

The concentration units are microgram cubic meter air ($\mu\text{g m}^{-3}$), $1 \mu\text{g m}^{-3} \sim 1$ molecule per billion air molecules

Challenge Time!

Add a plot in the ax subplot to visualise the measurements from both London and Bristol. The final plot should look as shown below:



Step 5: Data Analysis

✓ We can see that the NO₂ concentration has gone down in London starting in April, whereas Bristol measurements have been relatively constant over the four-month time frame. To quantify the change in NO₂ concentration we can average the concentration before and after the lockdown measurements were enforced by the UK government. The lockdown started on the evening of 23rd March and displaying the sf dataframe shows that this date corresponds to index 51.

```
In [32]: display(sf)
```

49	21/03/2020	21	V ugm-3	34	N ugm-3
50	22/03/2020	21	V ugm-3	31	N ugm-3
51	23/03/2020	37	V ugm-3	65	N ugm-3
52	24/03/2020	37	V ugm-3	49	N ugm-3
53	25/03/2020	50	V ugm-3	61	N ugm-3
54	26/03/2020	45	V ugm-3	32	N ugm-3
55	27/03/2020	33	V ugm-3	20	N ugm-3

✓ The 'London' array is split from index 51 into two separate arrays named 'London_prelock' and 'London_postlock'. The print command is then used to confirm the split.


```
In [61]: London_prelock=London[0:51]
London_postlock=London[51:len(London)]
print(London_prelock)
print(London_postlock)

[71. 68. 65. 44. 80. 84. 72. 65. 35. 69. 66. 74. 68. 77. 40. 55. 70. 74.
 76. 64. 74. 57. 40. 70. 86. 59. 54. 74. 61. 64. 61. 87. 90. 37. 62. 77.
 59. 79. 63. 75. 76. 69. 61. 53. 68. 66. 60. 29. 21. 34. 31.]
[65. 49. 61. 32. 20. 9. 7. 15. 24. 33. 35. 42. 32. 23. 36. 42. 48. 44.
 39. 37. 21. 4. 24. 46. 44. 21. 17. 11. 14. 15. 18. 30. 27. 21. 25. 31.
 19. 27. 27. 27. 20. 24. 22. 21. 28. 43. 39. 37. 6. 5. 17. 9. 15. 23.
 21. 23. 33. 29. 38. 37. 26. 15. 13. 23. 27. 32. 24. 29. 18. 12. 22.]
```

✓ We can use mean and std function of the numpy modules to take average and standard deviation values of the measurement values in the two split arrays. The standard deviation represents average fluctuation in the daily measurements.

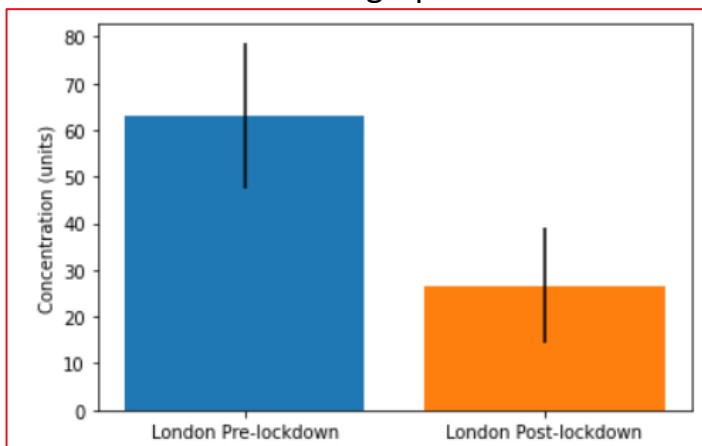
```
In [62]: London_prelock_mean=np.mean(London_prelock)
London_postlock_mean=np.mean(London_postlock)
London_prelock_std=np.std(London_prelock)
London_postlock_std=np.std(London_postlock)
print('London Prelockdown:', London_prelock_mean,London_prelock_std)
print('London Postlockdown:', London_postlock_mean,London_postlock_std)

London Prelockdown: 63.01960784313726 15.714973746278199
London Postlockdown: 26.661971830985916 12.43568656477135
```

✓ Finally, we can visualise these results in a bar plot to better summarise the results.

```
In [63]: fig=plt.figure()
ax = fig.add_subplot(111)
ax.bar('London Pre-lockdown',London_prelock_mean,yerr=London_prelock_std)
ax.bar('London Post-lockdown',London_postlock_mean,yerr=London_postlock_std)
plt.ylabel('Concentration (units)')
plt.show()
```

Shown below is the bar graph



✓ The black vertical lines in the bar plot represent the standard deviations in NO₂ measurements. The plot clearly shows that the mean NO₂ concentration has decreased after March 23. Moreover, even the upper limit value for the post lockdown measurements, considering fluctuations in daily measurements, never reach the lower limit value for the pre-lockdown measurements [i.e. the two black vertical lines do not overlap vertically]. Thus, we can safely conclude that the NO₂ measurement at the London site has decreased significantly due to the lockdown measures!

Science Alert!

The NO₂ concentration in London has decreased to within the recommended upper limit value of 40 $\mu\text{g m}^{-3}$ set by the World Health Organisation!

Challenge Time!

Find whether NO₂ concentration in Bristol has changed significantly due to the lockdown measures

Fun Challenge I!

Find which day of the week had the highest and lowest NO₂ concentrations on average in Bristol

Fun Challenge II!

Using the AirWeBreathe_seasonal.csv dataset (2019-2020 measurements) find whether the NO₂ concentration changes in Bristol and London are seasonal or due to the lockdown measures

This project guide was created by Dr. Rabi Chhantyal Pun from the School of Chemistry at University of Bristol as a part of the pathways to impact for the NERC grant NE/P013104/1