Introduction to Data Science

Ethan Meyers

2025-05-22

In September 2021, a significant jump in seismic activity on the island of La Palma (Canary Islands, Spain) signaled the start of a volcanic crisis that still continues at the time of writing. Earthquake data is continually collected and published by the Instituto Geográphico Nacional (IGN). ...

Table of contents

1 La Palma Earthquakes

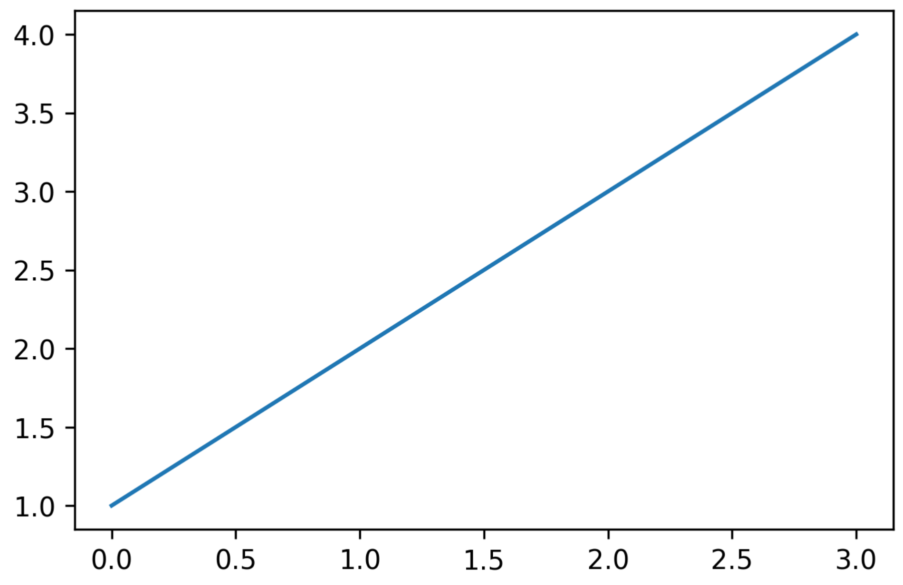
In September 2021, a significant jump in seismic activity on the island of La Palma (Canary Islands, Spain) signaled the start of a volcanic crisis that still continues at the time of writing. Earthquake data is continually collected and published by the Instituto Geográphico Nacional (IGN). ...

2 Testing!!! May 22 - update 2

2.1 Introduction

```python
import matplotlib.pyplot as plt
import numpy as np
eruptions = [1492, 1585, 1646, 1677, 1712, 1949, 1971, 2021]

plt.plot([1, 2, 3, 4])
```



```python
plt.figure(figsize=(6, 1))
plt.eventplot(eruptions, lineoffsets=0, linelengths=0.1, color='black')
plt.gca().axes.get_yaxis().set_visible(False)
plt.ylabel('')
plt.show()
```

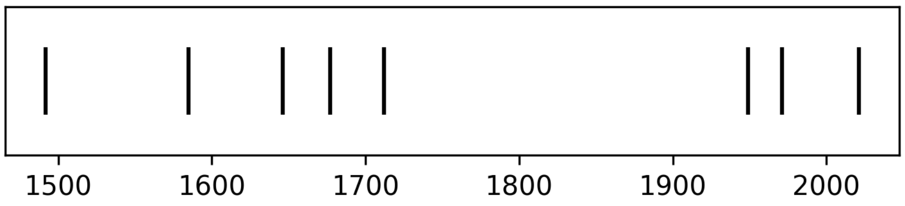


Figure 2.1: Timeline of recent earthquakes on La Palma

```
avg_years_between_eruptions = np.mean(np.diff(eruptions[:-1]))
avg_years_between_eruptions
```

Based on data up to and including 1971, eruptions on La Palma happen every 79.8 years on average.

Studies of the magma systems feeding the volcano, such as Marrero et al. (2019), have proposed that there are two main magma reservoirs feeding the Cumbre Vieja volcano; one in the mantle (30-40km depth) which charges and in turn feeds a shallower crustal reservoir (10-20km depth).

Eight eruptions have been recorded since the late 1400s (Figure **??**).

Data and methods are discussed in Section **??**.

Let x denote the number of eruptions in a year. Then, x can be modeled by a Poisson distribution

$$p(x) = \frac{e^{-\lambda}\lambda^{x}}{x!} \tag{2.1}$$

where λ is the rate of eruptions per year. Using Equation **??**, the probability of an eruption in the next t years can be calculated.

Table 2.1: Recent historic eruptions on La Palma

Name	Year
Current	2021
Teneguía	1971
Nambroque	1949
El Charco	1712
Volcán San Antonio	1677
Volcán San Martin	1646
Tajuya near El Paso	1585

Name	Year
Montaña Quemada	1492

Table **??** summarises the eruptions recorded since the colonization of the islands by Europeans in the late 1400s.

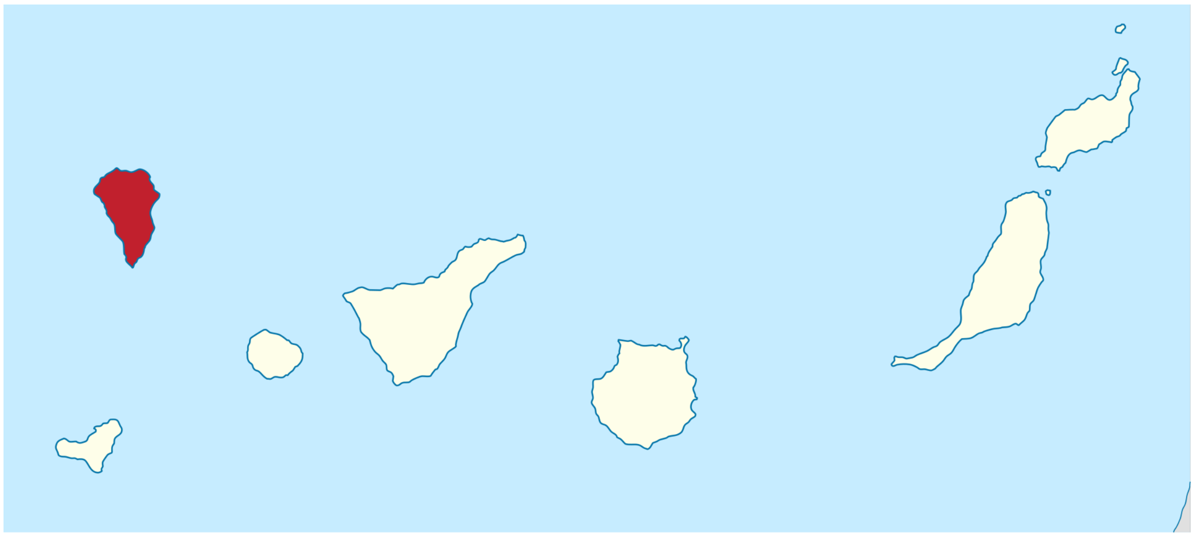


Figure 2.2: Map of La Palma

La Palma is one of the west most islands in the Volcanic Archipelago of the Canary Islands (Figure **??**).

Figure **??** shows the location of recent Earthquakes on La Palma.

2.2 Data & Methods

2.3 Conclusion

References

Marrero, José, Alicia García, Manuel Berrocoso, Ángeles Llinares, Antonio Rodríguez-Losada, and R. Ortiz. 2019. "Strategies for the Development of Volcanic Hazard Maps in Monogenetic Volcanic Fields: The Example of La Palma (Canary Islands)." *Journal of Applied Volcanology* 8 (July). https://doi.org/10.1186/s13617-019-0085-5.

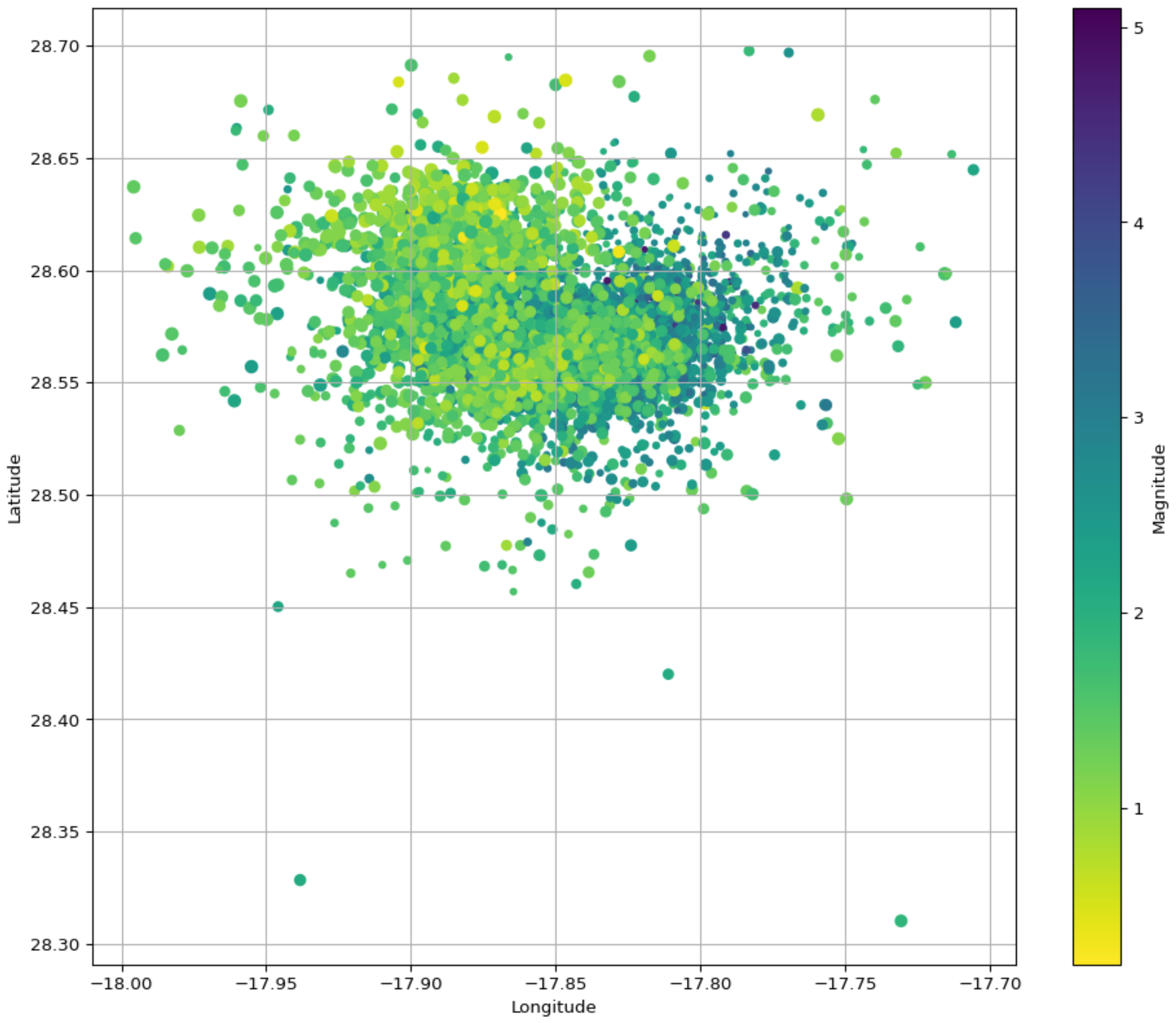


Figure 2.3: Locations of earthquakes on La Palma since 2017.

3 Python basics

This chapter introduces the fundamental concepts and basic syntax of the Python programming language. All the code covered here is part of the standard Python language and does not require any additional packages or libraries. Thus the Python discussed here forms the foundation for not only analyzing data in Python, but also for writing any kind of Python code.

While the chapter covers several different key concepts and syntax of Python, we focus on a subset of features that are most central for data analysis, rather than covering the full range of Python's capabilities. Becoming proficient in the basic Python covered in this chapter will be important as a basis for writing code in subsequent chapters, so make sure to practice and understand these concepts thoroughly.

By the end of this chapter, you should be comfortable with writing basic Python code, performing simple calculations, and understanding how Python represents and manipulates different types of data. These foundational skills will prepare you for more advanced topics in data analysis that are covered in the rest of the book.

3.1 Expressions

A **Python expression** is **any piece of code that produces a value.**. For example, the following is an expression that simply creates the number 21.

```
21
```

```
21
```

Similarly, an expression could be a series of mathematical operations that evaluate to number. For example, if want want to add 5 plus 2 and then multiple the result by 6 we can write:

```
6 * (5 + 2)
```

```
42
```

As mentioned above, the defining features of a *python expression* is that it produces a value. Expressions are one of the fundamental building blocks of data analysis and they will appear frequently throughout this book.

Exercise

What would happen if we remove the parenthesis from the expression we ran above and instead run `6 * 5 + 2`. See if you can predict what the result will be and then try it out in Python by running the code in a code cell and see if you get the result you predicted.

Solution

```
6 * 5 + 2
```

```
32
```

The result is 32, which makes sense because in the standard order of mathematical operations, multiplication occurs before addition so we multiple 6 * 5 and get 30, and then we add 2 to get 32.

3.1.1 Mathematical expressions

The expressions shown above were all "mathematical expressions" because they involve calculating numeric quantities. We can also write statements that will do operations on text and other types of data which we will describe more below. But first, let's explore mathematical expressions a bit more. Below is a table of some of the mathematical operations that are part of

Table 3.1: Python mathematical operators

Operation	Symbol	Example	Result
Addition	+	5 + 3	8
Subtraction	-	10 - 4	6
Multiplication	*	7 * 2	14
Division	/	12 / 5	2.4
Exponentiation	**	3 ** 2	9
Remainder	%	10 % 3	1

Exercise

What is the remainder from dividing 365 by 7? Please write some Python code that produces the answer.

Solution

```
365 % 7
```

```
1
```

3.2 Syntax

Syntax is the set of rules that defines how Python code **must** be written. One that think of syntax as the grammar of the Python programming language. In order for Python to be able to run your code, it **must** use the correct syntax.

To illustrate this, let's calculate the value of 8 squared (8^2) which hopefully you remember is equal to the value of 64. As shown Table **??**, if we want to take a value `x` to the power `y` (i.e., to calculate x^y) we use the syntax `x**y`. So, if we wanted to calculate 8^2 we would write the following Python code:

```
8**2
```

```
64
```

Since we have written the correct syntax, the code runs and the result of 64 is calculated as expected.

However, if we accidentially put an extra space between the two `*` symbols, Python will not know how to interpret the expression and we will get a syntax error as shown below:

```
8* *2
```

```
SyntaxError: invalid syntax (1271632921.py, line 1)
  Cell In[48], line 1
    8* *2
       ^
SyntaxError: invalid syntax
```

ERROR NUMBER 1: If python produces an error message with a ^ this will make it so that Quarto document will not be able to render to a pdf :(

For example, if I use a string without a closing quote this will produce this type of error

```
"hey
```

Likewise if I have a cell with the following it will produce this type of error:

```
"8 * * 2
```

When there is a syntax error, Python will print out `SyntaxError` and give you an indication where the syntax error has occurred using a ^ symbol.[1] As we can see here, Python is trying to show that the syntax error has occurred due to the extra space between the * symbols.

When there is a syntax error, Python will print out `SyntaxError` and give you an indication where the syntax error has occurred using a ^ symbol.[2] As we can see here, Python is trying to show that the syntax error has occurred due to the extra space between the * symbols.

The ability to be able to spot and fix syntax errors is a fundamental skill you will develop as become proficient in analyzing data in Python.

3.3 Assignment statements

An *assignment statement* is a line of code that is used to store a value in a named **variable**. We can then refer back to this variable name to retrieve the value we have stored.

To assign a value to a variable we use the `=` symbol. For example, the following code assigns the value `10` to the variable `a`:

```
a = 10
```

We can then refer back to the variable `a` later in our code to retrive the stored value. For example, if we just write `a` by itself on the last line of our Python code cell, it will print out the value stored in `a`.

```
a
```

```
10
```

[1] The reason this is a syntax error is because Python inteprets a single `*` symbol as a multiplicaiton symbol. Thus it is trying to multiple 8 by another multiplication symbol `*`, which gives an error since one can only multiply two numbers together.

[2] The reason this is a syntax error is because Python inteprets a single `*` symbol as a multiplicaiton symbol. Thus it is trying to multiple 8 by another multiplication symbol `*`, which gives an error since one can only multiply two numbers together.

As we can see, the value printed out is `10` which is the value we had previously stored in the name `a`.

If we were to assign the name `a` to another value, it will overwrite the previously stored value and `a` will store the new value.

```
a = 21
a
```

```
21
```

We an also do mathematical operations on values stored in variables, such as adding and multiplying variables together. For example, we can assign the variable `h` to store the value 24, and the variable `d` to store the value 7, and then we can multiple these together and store the result in the variable `t`.

```
h = 24
d = 7
t = h * d
t
```

```
168
```

Exercise

In the above code we calculated `t = h * d`. Which of the following do you think will happen to the value stored in `t` if we change the value of h to 3? I.e., if we run the following code, what do you think it will print out?

```
h = 3
t
```

a. The value of `t` will be change to be 21 (i.e., `7 * 3`).
b. The value of `t` will not change and will still contain `168`.
c. Something else will happen (e.g., Python will give an error).

Solution

```
h = 3
t
```

```
168
```

As you can see, the value of `t` did not change. This illustrates an important point that once a value is calculated and stored in a variable it will not change if the variable that were used as part of the calculation are updated!

3.3.1 Variable names

Variable names in Python must follow certain rules:

- Must start with a letter (a-z, A-Z) or an underscore (_), but not a number.
- Can contain letters, numbers, and underscores.
- Cannot contain spaces or special characters (like `@`, `#`, `$`, etc.).
- Cannot be a reserved Python keyword that are part of the Python language (like `for`, `if`, `class`, etc.).

If these rules are not followed, Python will produce a syntax error

It's also important to use meaningful variable names. For example, `t` is technically a valid variable name but it is not descriptive, while `total_hours` is much clearer. Using meaningful names makes your code easier to read and understand.

Exercise

The minimum wage in the United States in 2025 is $7.25. If someone works 40 hours per week for all 52 weeks in a year, what would there yearly earnings be? Please calculate by creating *meaningful* variable names for:

1. The minimum wage amount

2. The number of hours in a week

3. The number of weeks in a year

Then calculate the total yearly wage and store this result in another meaningful variable name, and print out the value stored in this last variable. Hint: Using underscores `_` in your variable names is highly encouraged to make them more readible.

Solution

```
hours = 24
days = 7
total_hours_in_a_week = hours * days
total_hours_in_a_week
```

```
168
```

3.4 Comments

Another very useful feature in Python is the ability to add **comments** to your code. Comments are lines in your code that are ignored by Python when your code runs. They are used to explain what your code is doing, make notes to yourself, or leave instructions for others who may read your code in the future.

In Python, you create a comment by starting the line with the # symbol. Anything after the # on that line will be treated as a comment and not executed.

For example:

```
# The code below calculates the number of seconds in a day
seconds_in_a_day = 60 * 60 * 24

seconds_in_a_day
```

```
86400
```

We will use comments extensively throughout this book to explain what code is doing and to make our code easier to understand. Adding clear comments is a good habit that will help both you and others who read your code in the future, so we strongly encourage you to add comments liberally for all code you write.

3.5 Functions (call expressions)

A **function** is a reusable piece of code that performs a specific task. You can think of a function as a "machine" that takes some input, does something with it, and then gives you an output.

Python comes with many built-in functions that you can use right away, and you can also load in additional functions in packages that other people have written. You can also write own functions, which is a topic we will discuss later in this book.

To use a function, you “call” it by writing its name followed by parentheses. If the function needs information to do its job, you put that information (called “arguments”) inside the parentheses.

For example, the `abs()` function take in a number and returns the absolute value of the number.

```python
abs(-10)
```

```
10
```

Some functions can take in multiple arguments. When multiple arguments are provided, they are separated by commas within the parentheses. For example, the `min()` function can take several numbers and will return the smallest one:

```python
min(10, 2, 87, 5, 90)
```

```
2
```

Another useful function is the `print()` function for displaying multiple pieces of information in a single Jupyter notebook code cell. By default, Jupyter will only display the result of the last line in a code cell. If you want to display multiple values or add custom messages, you can use the `print()` function.

For example, the code below will print both the numbers `2` and `3` in the same code cell. If we did not use the `print()` function, only the number `3` would be printed since it is the last line in the cell, but the number 2 would not be printed because it is on the last line in the cell.

```python
# We need to call print() explicitly here to print the value of
# 2 since it is not on the last line of the code cell

print(2)



# The value of 3 will be printed here without needing to call
# the print() function because it is the last line in the cell

3
```

```
2
```

```
3
```

Exercise

Try using the `print()` function to display both a message and a value in the same output. For example, print the message "The answer is:" followed by the result of `6 * 7`.

Solution

```python
print("The answer is:")
6 * 7
```

```
The answer is:
```

```
42
```

```python
# We can also print multiple pieces of text on a single line by
# passing multiple arguments to the print() function:

print("The answer is:", 6 * 7)
```

```
The answer is: 42
```

3.6 Data types

Python is able to process many different types of data, referred to as "data types". So, far we have only explored numeric data. Let's continue exploring numerical data in a little more detial and then we will go on to examine other types of data.

3.6.1 Numbers

Python uses two different formats to store numerical data known as "integers" and "floating-point numbers".

- **Integers** (`int`): Whole numbers without a decimal point, such as `5`, `-3`, or `1000`.
- **Floating-point numbers** (`float`): Numbers that have a decimal point, such as `3.14`, `-0.5`, or `2.0`.

We can tell if a number is a floating point number (i.e., a "float") by seeing if there is a decimal point at the end of the number when we print out the number.

```
# This is an integer, which we can tell becaues there is no decimal point
5
```

```
5
```

```
# Although we are dividing two integers, the result is a floating point number
# which we can tell becaues there is a decimal point

10/2
```

```
5.0
```

We can also use the `type()` function to check if a number is an integer or a floating point number.

```
# This is a floating point number

type(5.0)
```

```
float
```

When analyzing the data, usually it does not matter if Python is storing a number as an integer or a floating point number since Python does the math sensibly and converts between integers and floating point numbers as needed. However, internally Python is representing these number is quite different ways.

More imporantly is to know that there are some limitations to the way Python stores both integers and floats. In particular, both of these types of numbers are represented using a finite amount of memory, so there is a largest number integer that can be represented and a limit to the precision of floating-point numbers. For most practical purposes, these limits are very large, but you may encounter issues with extremely large numbers or with floating-point arithmetic where results are not exactly as expected due to rounding errors.

For example, if we multiple intergers that are too long, we can get a `ValueError` which indicates that Python is running into problems representing an integer this large.