Introduction to Data Science

Ethan Meyers

Table of contents

Welcome

This book gives an introduction to Data Science using the Python programming langauge.

1 Introduction

In this chapter we will discuss what the field of Data Science is, and give a brief history of how the field developed.

This book is your guide to understanding the exciting and increasingly influential field of data science. Whether you're curious about how data shapes our world or are looking to explore the possibilities of data-driven insights, this book will provide you with a foundational understanding of what data science is and why it matters.

1.1 What is Data Science?

Data science is a dynamic and interdisciplinary field that combines techniques and theories from statistics, computer science, and specialized knowledge in various areas to extract valuable knowledge and insights from data [Chapter 1]. This data can come in many forms, whether neatly organized in databases or existing as unstructured information like text or images.

At its core, data science follows a systematic process for analyzing data. This includes a range of crucial steps, starting with data collection and ensuring the data is in a usable state through data cleaning. Once prepared, the data is explored to uncover initial patterns and relationships (data exploration). Data scientists then apply various modeling techniques to identify deeper insights, which need to be carefully interpreted to draw meaningful conclusions. Finally, the findings are communicated effectively to inform decisions and understanding [Chapter 1].

The field of data science has experienced remarkable growth in recent years This surge in prominence can be attributed to several key factors: - The explosion in the amount of data being generated across all sectors, from social media to scientific research. - Significant advancements in computing power, enabling the processing and analysis of these vast datasets. - The development of increasingly sophisticated analytical tools and techniques that allow for more complex and insightful data exploration.

By delving into data science, you can gain practical analytical skills that are applicable across a wide array of fields [Chapter 1, 62]. You'll learn how to approach real-world data, identify key questions, and use data-driven methods to find answers and understand the world around us [Chapter 1, 62]. As a lighthearted starting point, you might hear the quip that "A Data Scientist is a Statistician who lives in San Francisco" [Chapter 1, 11]. While humorous, this

simple definition hints at the combination of statistical thinking with the technological innovation often associated with data science. Throughout this book, we will move beyond simplistic definitions to explore the rich and multifaceted nature of this vital field.

Key points - Despite the fact that humans have been collecting data for millenia, and doing sophisticated analyses of data for centuries, the field of data science" (or at least the name) is relatively new. -

1.2 A brief history of Data Science

1.2.1 A brief history of data

1.2.2 A brief history of Statistics

1.2.3 A brief history of computation

Computational devices also have a long history.

1.2.4 The creation of the field of Data Science

2 Python basics

Now that we have discussed what data science is, let's learn some of the basics of the Python programming language. Once we have learned some of these Python basics we can begin to analyze data!

2.1 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.

Answer:

```
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.

2.1.1 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 2.1: 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.

Answer:

```
365 % 7
```

```
1
```

2.2 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. If incorrect syntax is used, then one will get a "syntax error", and the code will not run.

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 2.1, 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 (3783800731.py, line 1)
  Cell In[6], line 1
    8* *2
       ^
SyntaxError: invalid syntax
```

When there is a syntax error, Python will print out `SyntaxError` and give you an indication where the syntax error has occurred using a caret symbol (should replace "caret" with the actual symbol but this is causing an error when compiling the book on GithHub). 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.

2.3 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
```

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).

Answer

```
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!

2.4 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, and 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 name. Hint: Using underscores `_` in your variable names is highly encouraged to make them more readible.

Answer

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

```
168
```

2.5 Functions (call expressions)

Could do functions first before donig data types. That way I can use the `type()` function and the `int()` and `float()` functions. Could then go into more detail with functions on strings (perhaps in the string section). OK, do it!

2.6 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.

2.6.1 Numbers

Python uses two different formats to stores numerica 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.

When analyzing the data, usually it does not matter if Python is storing a number as an integer or a floating point number, so usually it will not affect our code or our analyses. However, under the hood, Python is actually 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. First, and most obviously,

2.6.2 Character strings

2.6.3 Booleans

You can check the type of a value in Python using the `type()` function:

2.7 Lists

2.8 Other data structures

2.8.1 Tuples

2.8.2 Dictionaries

3 Array computations

This is a book created from markdown and executable code.

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

```
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()
```

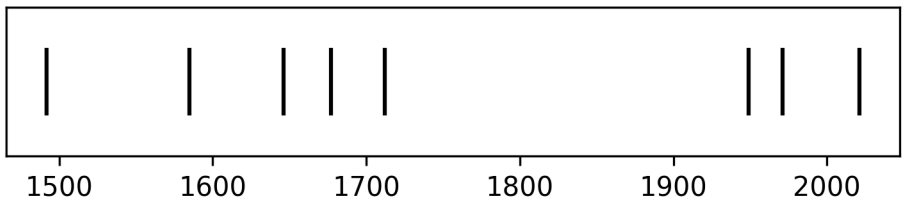

Figure 3.1: Timeline of recent earthquakes on La Palma

```
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 3.1).

Data and methods are discussed in Section 3.1.

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

where λ is the rate of eruptions per year.

Table 3.1: 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
Montaña Quemada	1492

Table 3.1 summarises the eruptions recorded since the colonization of the islands by Europeans in the late 1400s.

Figure 3.2 shows the location of recent Earthquakes on La Palma.

3.1 Data & Methods

3.2 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.

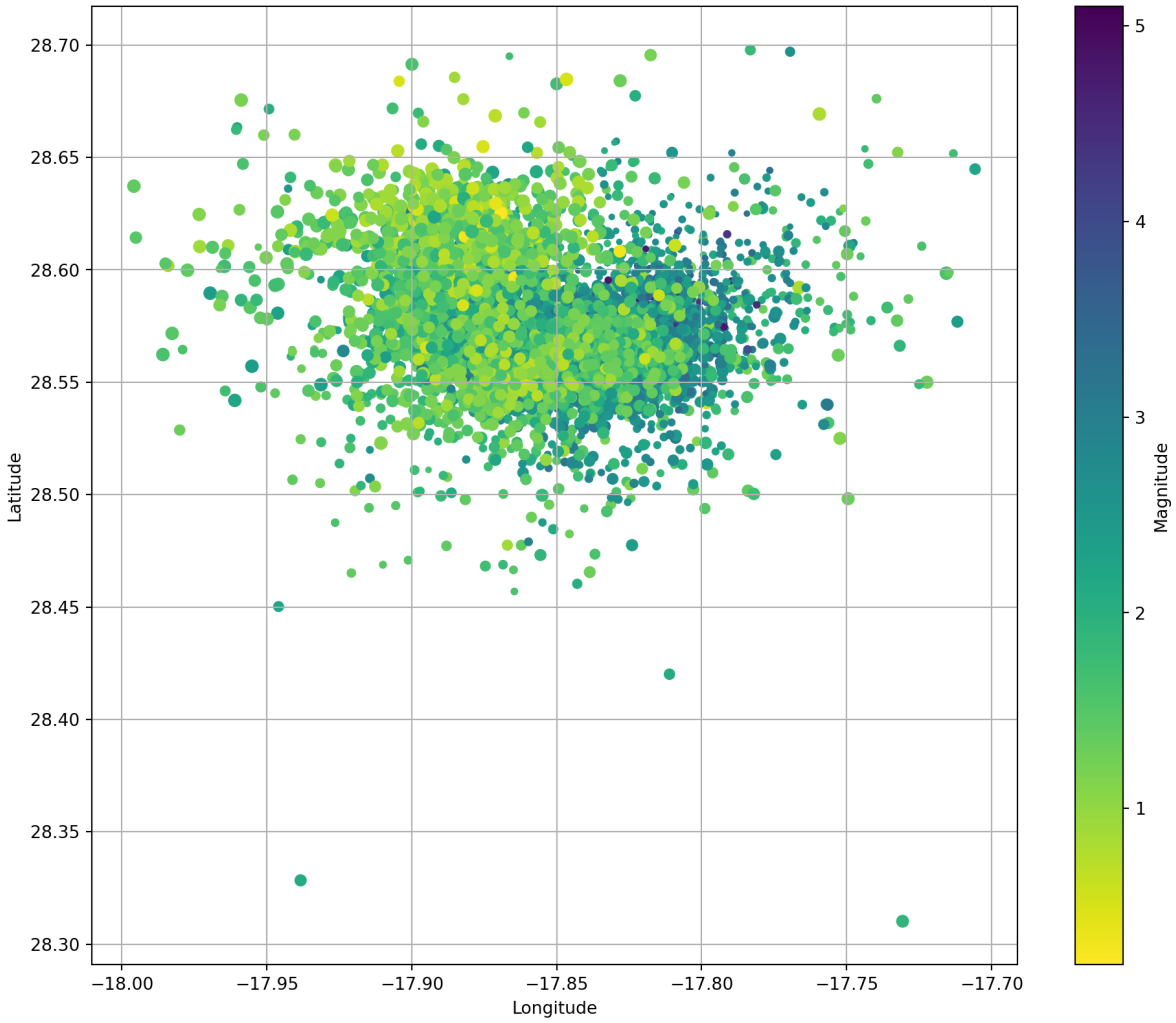

Figure 3.2: Locations of earthquakes on La Palma since 2017.

4 Data tables

5 Data visualization