CHAPTER 2: DATA STRUCTURES

DR. MOHD KHAIRUL BAZLI BIN MOHD AZIZ

PUSAT SAINS MATEMATIK, UNIVERSITI MALAYSIA PAHANG

OVERVIEW

This chapter will introduce advanced data operations on built-in data structures. You can utilize these data structures to solve data-wrangling problems. After reading this chapter, you will be able to compare Python's advanced data structures and make use of the Operating System (OS) file-handling operations. This chapter focuses on the data structures in Python and the OS functions that are the foundation of this book. By the end of this chapter, you will have learned how to handle advanced data structures.

CONTENT

2.1 Describing Data

2.2 Understanding data structures

2.3 Data Structures and Sequences

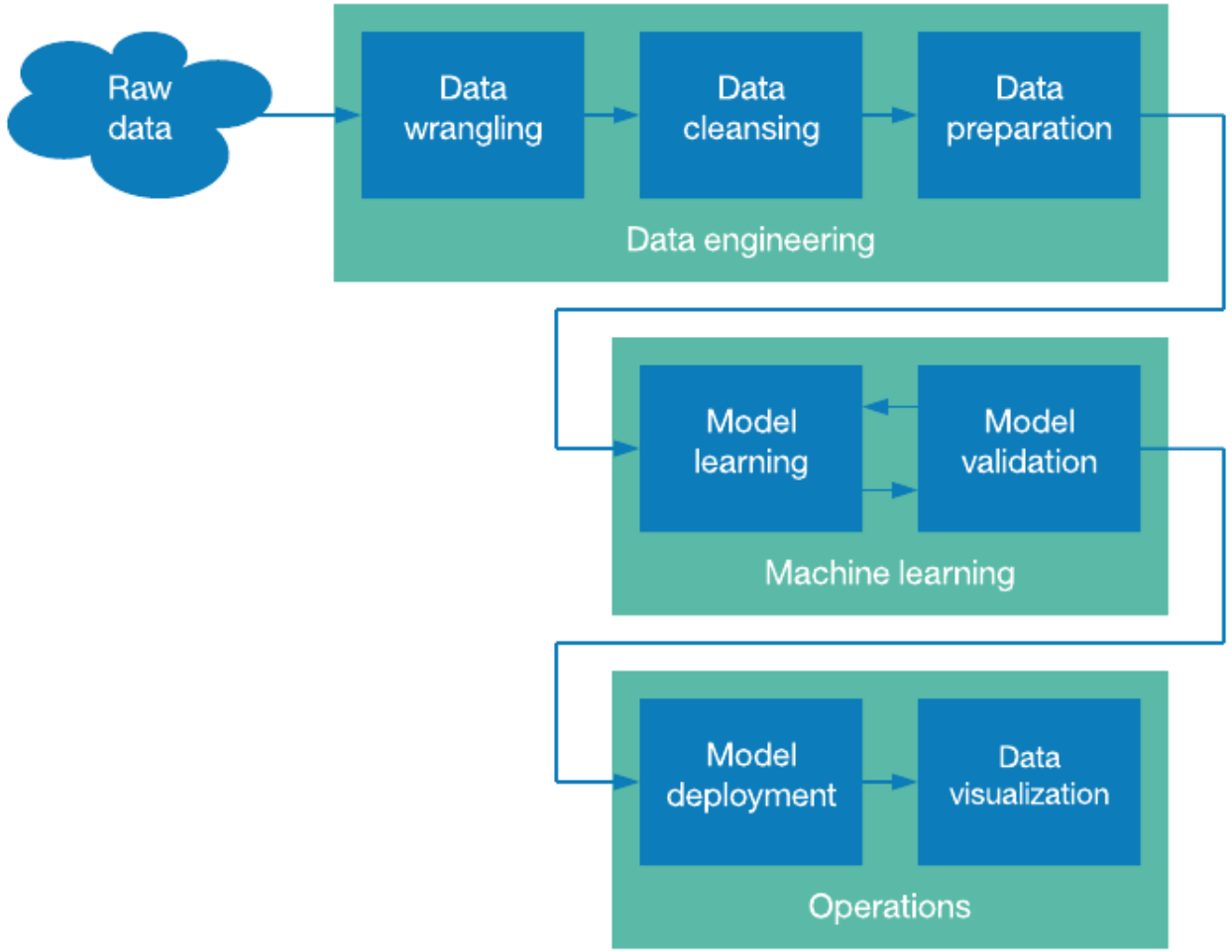
2.4 Functions

2.1 DESCRIBING DATA

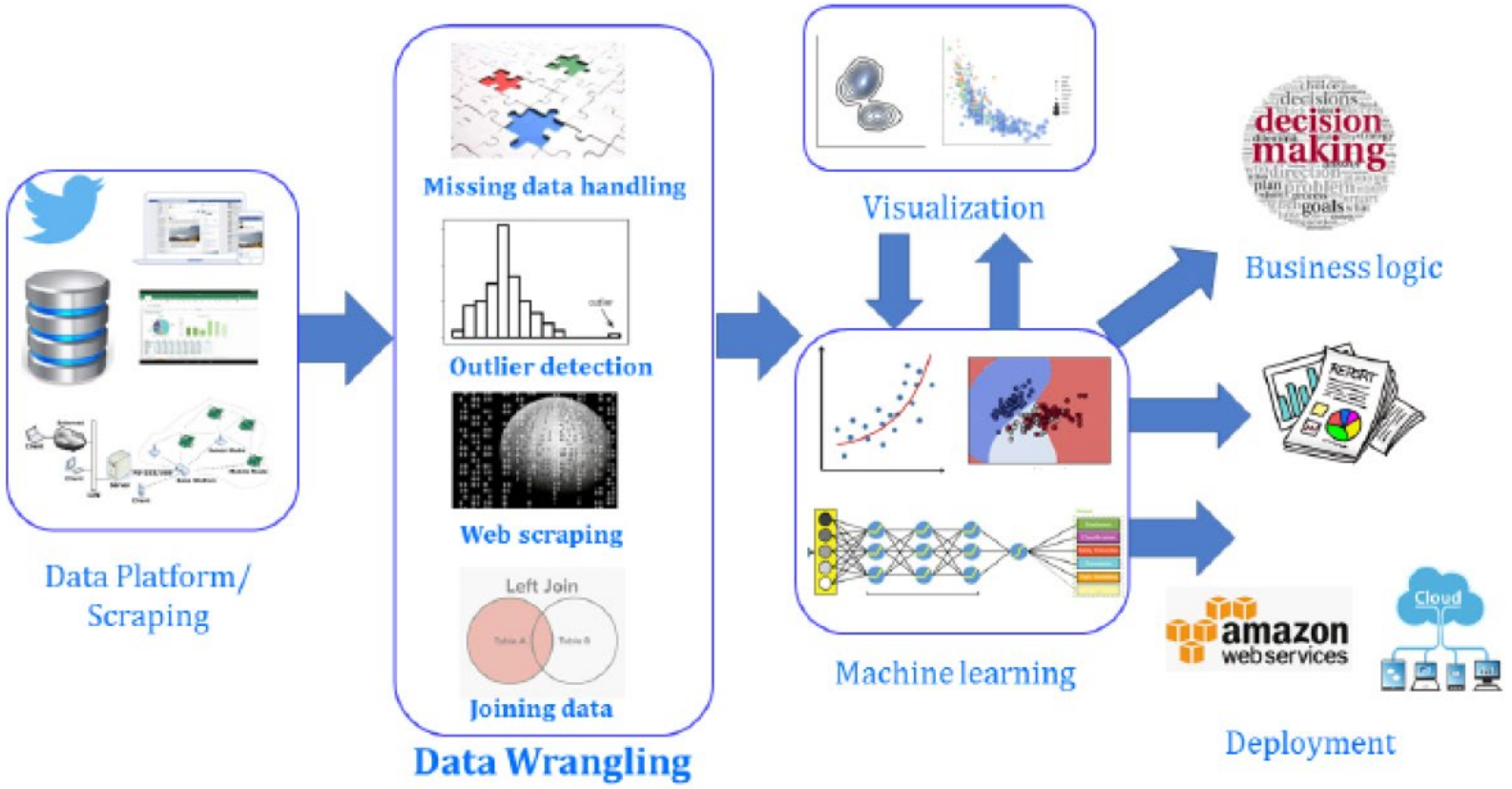
DATA

- Data is a commodity, but without ways to process it, its value is questionable. Data science is a multidisciplinary field whose goal is to extract value from data in all its forms. This article explores the field of data science through data and its structure as well as the high-level process that you can use to transform data into value.
- Data science is a process. That's not to say it's mechanical and void of creativity. But, when you dig into the stages of processing data, from munging data sources and data cleansing to machine learning and eventually visualization, you see that unique steps are involved in transforming raw data into insight.

THE DATA SCIENCE PIPELINE



PROCESS OF DATA WRANGLING



CREATING METADATA

- Datasets are composed of records. Records are composed of fields. Records often represent or correspond to people, objects, relationships, or events. The fields within a record represent or correspond to measurable aspects of the person, object, relationship, or event.
- When you are describing your data, you should be focused on understanding the structure, granularity, accuracy, temporality, and scope of your data. Structure, granularity, accuracy, time, and scope are key aspects of representational consistency. As such, they are also the characteristics of a dataset that must be tuned or improved by your wrangling efforts.

STRUCTURE

- The structure of a dataset refers to the format and encoding of its records and fields.
- Assessing the structure of your dataset is primarily a generic metadata question.
- Here are some of the questions you need to ask when assessing data structure:
  1. Do all records in the dataset contain the same fields?
  2. How can you access the same fields across records? By position? By name?
  3. How are the records delimited/separated in the dataset? Do you need sophisticated parsing logic to separate the records from one another?
  4. How are the record fields delimited from one another? Do you need to parse them?
  5. How are record fields encoded? Human readable strings? Binary numbers? Hash keys? Compressed? Enumerated codes?

GRANULARITY

- The granularity of a dataset refers to the kinds of entities that each data record represents or contains information about. In their most common form, records in a dataset will contain information about many instances of the same kind of entity.
- For example, a dataset in which a single record represents a single sales transaction by a single customer at a particular store would have a fine granularity.
- A dataset in which each record represents the total sales in each store for each day would have a coarse granularity. At an even coarser granularity, you might have a dataset in which each record represents total sales by region and week.

ACCURACY

- The accuracy of a dataset refers to its quality. In other words, the values populating record fields in the dataset should be consistent and accurate. For example, consider a customer actions dataset. This dataset contains records corresponding to when customers added items to their shopping carts.
- Common inaccuracies are misspellings of categorical variables, like street or company names; lack of appropriate categories, like ethnicity labels for multiethnic people; underflow and overflow of numerical values; and missing field components, like a timestamp encoded in a 12-hour format but missing an AM/PM indication.

TEMPORALITY

- A data record is a representation of an entity at a particular time (or set of times).
- Accordingly, even though a dataset might have been an accurate and consistent representation at the time it was created, subsequent changes might render the representation inaccurate or inconsistent.
- The time-sensitive nature of representations, and hence of datasets, is an important aspect that should be explicitly noted.

VALUE

Indirect value

- Data provides value to your organization by influencing people's decisions or inspiring changes in processes. Example: risk modeling in the insurance industry.

Direct value

- Data provides value to your organization by feeding automated systems. Example: Netflix's recommendation system.

SCOPE

- The scope of a dataset has two major dimensions.
- The first dimension concerns the number of distinct attributes represented in a dataset. For example, for each customer action, we might know when it happened (e.g., a timestamp) and some details about it (like which UPC a customer added to a basket).
- The second dimension concerns the attribute-by-attribute population coverage: are "all" the attributes for each field represented in the dataset, or have some been randomly, intentionally, or systematically excluded?

2.2 UNDERSTANDING DATA STRUCTURES

DATA AND ITS STRUCTURE

- Data comes in many forms, but at a high level, it falls into three categories: structured, semi-structured, and unstructured.
- Structured data is highly organized data that exists within a repository such as a database (or a comma-separated values [CSV] file). The data is easily accessible, and the format of the data makes it appropriate for queries and computation (by using languages such as Structured Query Language (SQL) or Apache™ Hive™).
- Unstructured data lacks any content structure at all (for example, an audio stream or natural language text).
- In the middle is semi-structure data, which can include metadata or data that can be more easily processed than unstructured data by using semantic tagging. This data is not fully structured because the lowest-level contents might still represent data that requires some processing to be useful.

MODELS OF DATA

Structured data	Semi-structured data	Unstructured data
Databases	XML / JSON data Email Web pages	Audio Video Image data Natural language Documents

DATA AND ITS STRUCTURE

- Structured data is the most useful form of data because it can be immediately manipulated. The rule-of-thumb is that structured data represents only 20% of total data. Most of the data in the world (80% of available data) is unstructured or semi-structured.
- Note that much of what is defined as unstructured data actually has structure (such as a document that has metadata and tags for the content), but the content itself lacks structure and is not immediately usable. Therefore, it is considered unstructured.

2.3 DATA STRUCTURES AND SEQUENCES

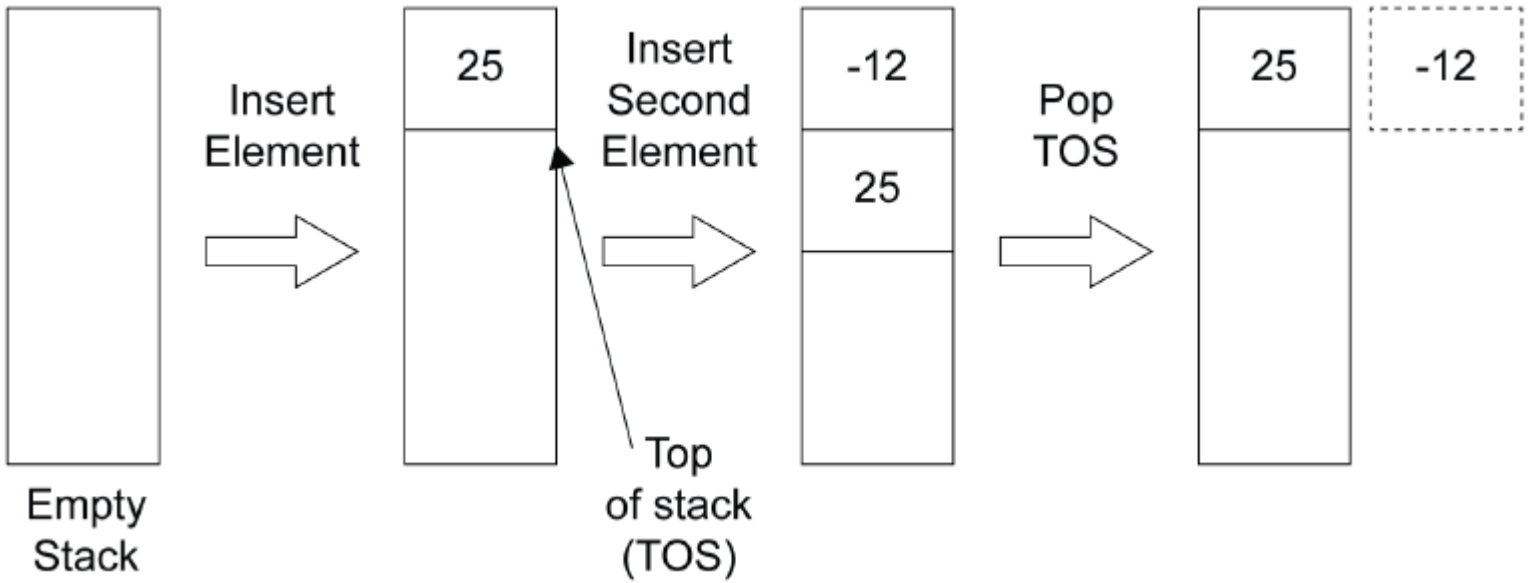
- List
- Sets
- Strings
- Tuples
- Dictionaries

2.4 FUNCTIONS

ITERATOR

- Iterators in Python are very useful when dealing with data as they allow you to parse the data one unit at a time. Iterators are stateful, which means it will be helpful to keep track of the previous state. An iterator is an object that implements the **`next`** method—meaning an iterator can iterate over collections such as lists, tuples, dictionaries, and more. Practically, this means that each time we call the method, it gives us the next element from the collection; if there is no further element in the list, then it raises a **`StopIteration`** exception.

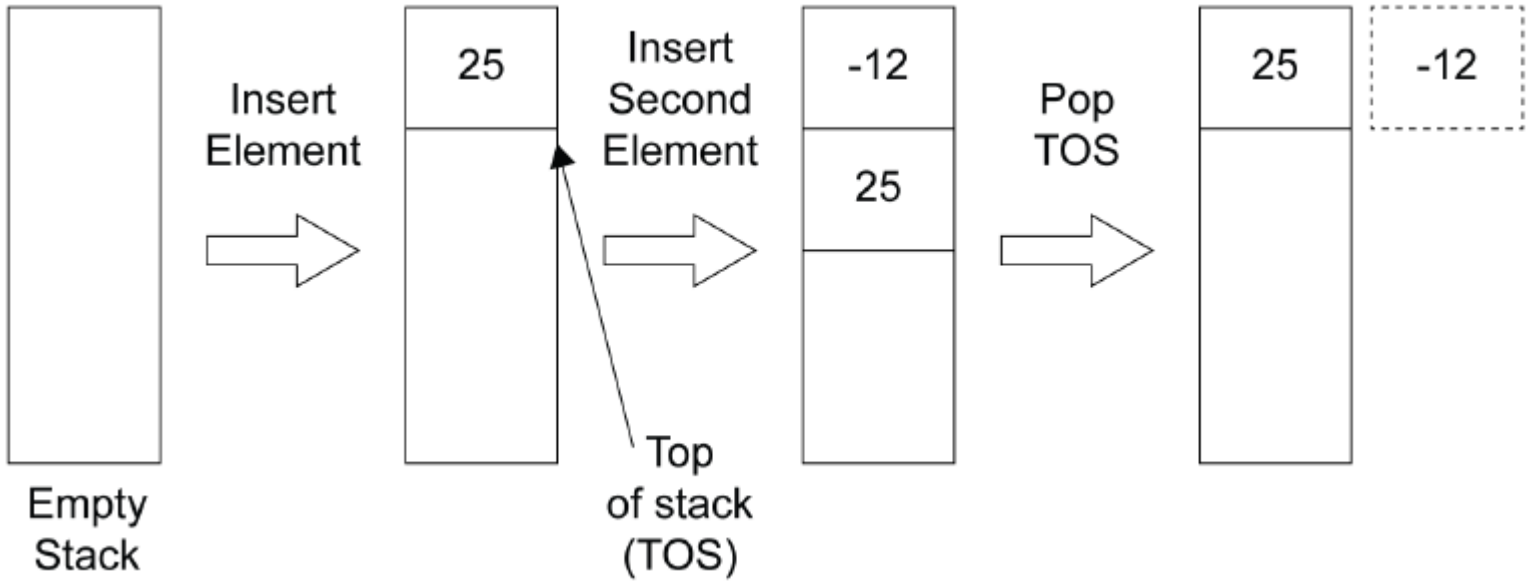
STACKS



- A stack is a very useful data structure. If you know a bit about CPU internals and how a program gets executed, then you will know that a stack is present in many such cases. It is simply a list with one restriction, **Last In First Out** (**LIFO**), meaning an element that comes in last goes out first when a value is read from a stack. The following illustration will make this a bit clearer:

STACKS

- As you can see, we have a LIFO strategy to read values from a stack. We will implement a stack using a Python list. Python lists have a method called **`pop`**, which does the exact same **`pop`** operation that you can see in the preceding illustration. Basically, the **`pop`** function will take an element off the stack, using the **Last in First Out** (**LIFO**) rules. We will use that to implement a stack in the following exercise.

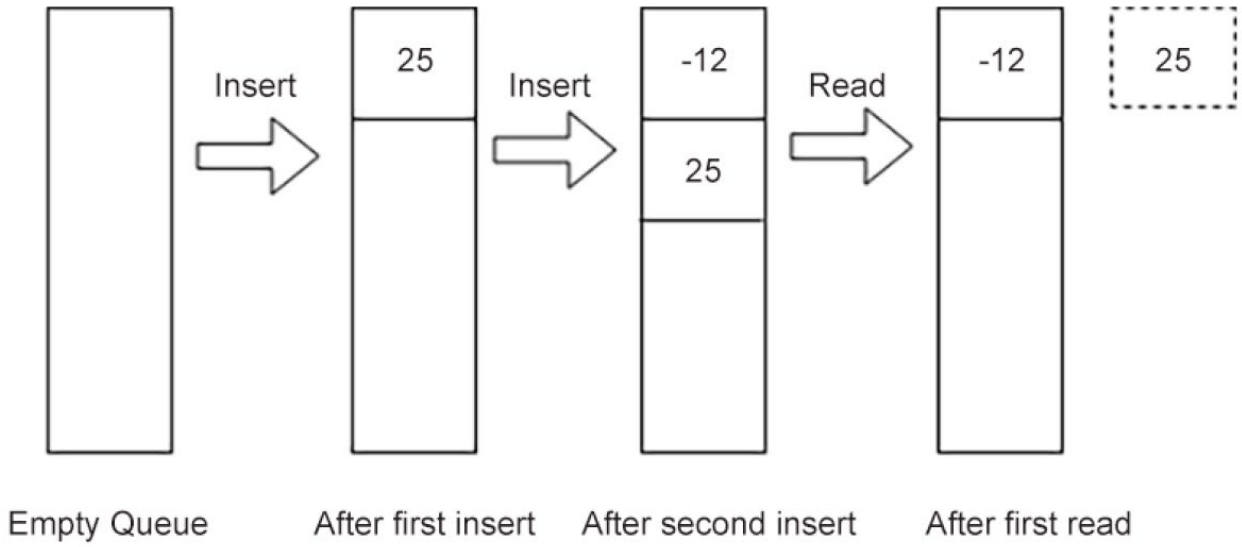


LAMBDA EXPRESSIONS

- In general, it is not a good idea to change a variable's value inside a function. Any variable that is passed to the function should be considered and treated as immutable. This is close to the principles of functional programming. However, in that case, we could use unnamed functions that are neither immutable nor mutable and are typically not stored in a variable. Such an expression or function, called a **lambda expression** in Python, is a way to construct one-line, nameless functions that are, by convention, side-effect-free and are loosely considered as implementing functional programming.

QUEUE

- Apart from stacks, another high-level data structure type that we are interested in is queues. A queue is like a stack, which means that you continue adding elements one by one. With a queue, the reading of elements obeys the **First in First Out (FIFO)** strategy. Check out the following diagram to understand this better:



しつもん が ありますか

ありがとう ございます