Lecture 1

Introduction: Data Science for Chemistry

Contents

Chemical data science applies data science methods to chemical problems. Accordingly, this course is organized around two streams: data science foundations and chemical applications. The course introduces core data science concepts and techniques, accompanied by hands-on exercises demonstrating how these methods can be applied to real chemical problems. The topics covered in this course include:

- Data science
  - Data structures, acquisition, cleaning, visualization, feature engineering, analysis, storage, and publication.
  - * Statistics, machine learning, and artificial intelligence.
  - * Programming modules, software packages, and online resources.
- Chemical applications
  - Chemical data acquisition, data representation, and modeling chemical problems.
  - Specific applications: property prediction, chemical reactions, spectroscopy, experimental pipelines, and chemical design, etc.

Since this course encompasses a broad range of topics, not all aspects can be covered in detail, particularly those marked with an asterisk (*). The course will emphasize high-level theoretical understanding rather than detailed mathematical derivations and formal proofs. Instruction will

focus on the effective use of existing programming modules and software packages, rather than on implementing algorithms from scratch.

By the end of the course, students should be familiar with major data science methods and gain experience applying them to chemical research problems.

1 Basics about Data

According to Wikipedia, "data are a collection of discrete or continuous *values that convey information*, describing quantity, quality, facts, statistics, or other basic units of meaning, or simply sequences of symbols that may be further interpreted formally." Data have no intrinsic meaning until they are interpreted or analyzed. For example, the grades of students in a class are data; only when these grades are analyzed can we assess student performance.

Data can take many forms. In a sense, each of us can be considered a datum in the context of a larger dataset. To process and communicate data efficiently, we adopt *standardized forms* that can be stored and processed digitally. In this course, we focus on such digital data representations.

Several fundamental concepts are important to understand before working with data:

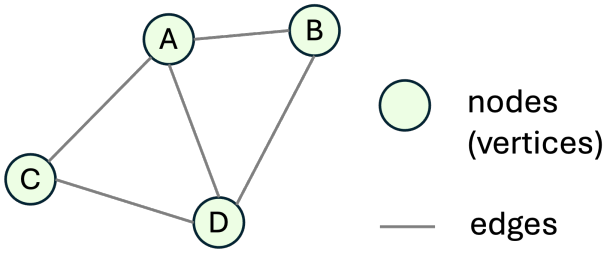
- **Data type.** The fundamental building blocks in programming and data analysis. Examples include integers, floating-point numbers, strings, and boolean values.
  Examples: In Python, `5` is an integer (`int`), `5.0` is a floating-point number (`float`), and `'5'` is a string (`str`).
  In Python, data type is handled *implicitly*, you can declare values simply by

  ```
  a = 5
  b = '5'
  ```

  However, if you type `a + b`, you will get error

  ```
  TypeError: unsupported operand type(s) for +: 'int' and 'str'
  ```

- **Data structure.** The organization of data in memory or storage to allow efficient access and modification. Common structures include lists, arrays, dictionaries, and more complex forms like trees or graphs.
  Examples
  - **Lists**: `[0, 1, 2, 3]`, `[0, 'a', 1.2, True]`.
  - **Dictionaries**:commonly used in data storage.
    `{'a': 1, 'b': 2}`,
    `{'name': 'a list', 'list_val': [0,1,2]}`
  - **Graphs**

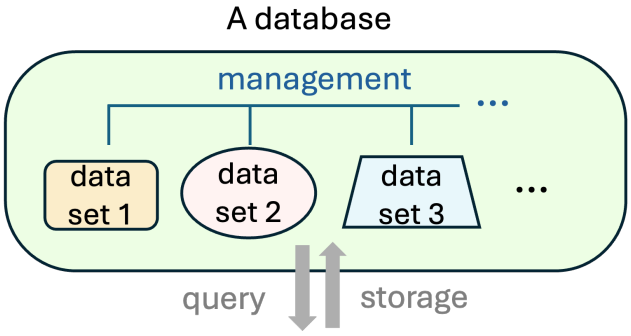


You can represent the graph with a *disctionary*:

```
{'A': ['B', 'C', 'D'],  'B': ['A', 'D'],
'C': ['A', 'D'],  'D': ['A', 'B', 'C']}
```

Molecules can be seen as graphs: atoms as nodes and bonds as edges.

- **Dataset.** A collection of related data, typically organized in a tabular or structured form. For example, a CSV file recording students' grades in different subjects constitutes a dataset.
- **Database.** A systematic collection of datasets stored for long-term management, often supporting queries, updates, and concurrent access. Databases can be relational (tables with relationships), document-based (JSON or XML), or other specialized forms.
  An illustration of database is shown below



- **Metadata.** Data that describes other data. For instance, in a student grades dataset, metadata could include column names, data types, grading scale, or the date of collection. Metadata is essential for interpreting the dataset correctly.
- **Data schema.** A formal description of how data is structured, organized, and related, including the types of data, relationships among elements, and any constraints or rules. It acts as a *blueprint* for understanding, storing, validating, and analyzing data.

The concept of a data schema can be abstract. To illustrate, consider the following examples of data points:

```
compound_name: Benzene
CAS: 71-43-2
wavelength_nm: 200
absorbance: 0.12
solvent: Hexane
```

(a) A flat data structure.

```
name: Benzene
CAS: 71-43-2
solvent: Hexane
spectrum:
type: UV-Vis
points:
  - wavelength_nm: 200
  - absorbance: 0.12
```

(b) A hierarchical data structure.

Figure 1: Examples of flat and hierarchical data structures.

The left example represents a flat data structure. A corresponding data schema is shown in Table 1.

Table 1: Data schema for a flat UV–Vis dataset

Column Name	**Data Type**	**Description**
compound_name	string	Name of the chemical (e.g., Benzene)
CAS	string	CAS Registry Number
wavelength_nm	float	Wavelength of measurement (nm)
absorbance	float	Measured absorbance at the wavelength
solvent	string	Solvent used for measurement (optional)

In most chemical data applications, we work with hierarchical data structures, as in the right example in Fig. 1. Here, the ‘compound‘ category encloses all related information, and ‘spectrum‘ contains subcategories for type and measurement points. A schema for this hierarchical dataset is shown in Table 2.

Table 2: Data schema for a hierarchical UV–Vis spectrum

Component	**Data Type**	**Description**
name	string	Chemical name (e.g., Benzene)
CAS	string	CAS Registry Number
solvent	string	Solvent used in measurement (optional)
spectrum.type	string	Spectroscopy type (UV–Vis)
spectrum.points.wavelength	float	Wavelength of each measurement (nm)
spectrum.points.absorbance	float	Measured absorbance at each wavelength

Note that not all databases provide a separate schema description. Often, you need to examine the data files themselves to infer the schema.

2 Python Basics

Python will be used as the primary programming language throughout this course. There are several ways to run Python programs. For large-scale simulations or memory-intensive tasks, scripts should be executed on a high-performance cluster or a sufficiently powerful desktop. For lightweight scripts, exploratory analysis, and teaching purposes, Python notebooks provide a more convenient and interactive environment.

2.1 Python Notebooks

In this course, we will primarily use cloud-based Python notebooks. Two platforms will be used:

- **Google Colaboratory (Colab).**
  If you have a Google account, Colab can be accessed through Google Drive or directly at `https://colab.research.google.com/`. YColab is well suited for small to moderate computations and is an excellent platform for sharing and reproducing code.
- **Vocareum.**
  Vocareum is a hands-on lab platform specializing in cloud, data science, and AI education, with access to high-end GPU resources. This course is supported by Vocareum, and students will have access to it throughout the semester.

Python notebooks are organized into *cells*. Each cell can be executed independently, allowing code to be run incrementally rather than as a single script. To execute a cell, click the run button (▶ Run) or press **Shift + Enter**. To execute the cell and create a new one below, press **Alt + Enter**.

2.2 Terminal access

For large-scale simulations, students may wish to use the Amarel high-performance computing cluster. Instructions for setting up an Amarel account can be found at `https://sites.google.com/view/cluster-user-guide/amarel?authuser=0`.

Note: Terminal usage and cluster workflows will not be covered in this course, as it is not a programming or HPC-focused class. However, additional guidance can be provided upon request.

2.3 Basic Python Syntax

Python is designed to be readable and beginner-friendly. This section introduces a few essential concepts and commands needed to get started.

1. **Installing packages.**
   Python has a large ecosystem of third-party packages that can be installed to extend its functionality. In a notebook, packages can be installed using:

```
!pip install [package-name]
```

Note that the `!` symbol indicates that the command is executed in the system shell rather than as Python code. On Vocareum, many commonly used packages are pre-installed.

2. **Importing packages.**
   To import an entire package:

   ```
   import [package-name]
   ```

   To import a specific module or function:

   ```
   from [package-name] import [module-name]
   ```

3. **Calling functions**.
   Functions are invoked using parentheses:

   ```
   [result] = [function-name]([arg1], [arg2], ...)
   ```

   If the function belongs to a module, use:

   ```
   [result] = [module-name].[function-name]([arg1], [arg2], ...)
   ```

4. **For loops.**
   A `for` loop allows a block of code to be executed repeatedly over a sequence of values:

   ```python
   for i in [0, 1]:
       print(i)
   ```

   The colon (`:`) and indentation define the scope of the loop. This syntax pattern appears frequently in Python, including when defining functions and conditional statements.

The following example combines these concepts:

```python
import math

def cos_sin(x):
    return math.cos(x) + math.sin(x)

for x in [0, 1, 2]:
    y = cos_sin(x)
    print(x, y)
```

As the course progresses, you will become increasingly comfortable reading and writing Python code in this style!

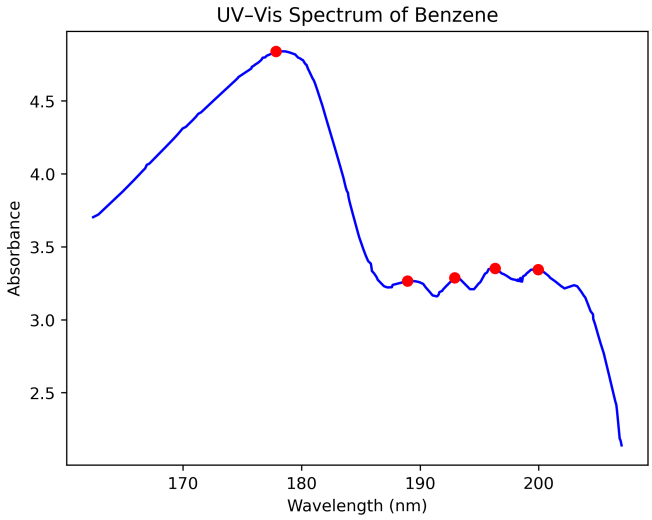


Figure 2: UV–Vis spectrum for benzene.

3 Lab 0: Python Basics

More details about Python basics is provided in Lab 0: `https://colab.research.google.com/drive/1qX5824QN5e5PuFbRNiDP_kQ5slC729oa?usp=drive_link`

4 Lab 1: UV–Vis Spectra

UV–Vis spectroscopy measures the absorption of ultraviolet and visible light by a chemical substance. The primary measured quantity is the *absorbance* (or, in some cases, transmittance) as a function of *wavelength*.

In this lab, we focus on using UV–Vis spectra as an example to practice fundamental data science skills. You will learn how to load spectral data stored in a CSV file, perform basic data cleaning, visualize the spectrum, and identify absorbance peaks. These steps mirror common tasks encountered when working with experimental or database-derived scientific data.

While UV–Vis spectroscopy can provide insight into electronic excitations in molecules, our emphasis in this lab is on data handling and analysis rather than detailed spectroscopic interpretation. Students interested in learning more about the chemical principles underlying UV–Vis spectra may consult the following reference: `https://www2.chemistry.msu.edu/faculty/reusch/virttxtjml/spectrpy/uv-vis/spectrum.htm`

We will cover the following contents:

- Python packages:

- `pandas`: loading, analyzing and saving tabular data such as `.csv` (comma-separated values) files.
- `matplotlib`: visualize scientific data.

We will reproduce the UV–Vis spectrum of benzene shown in Fig. 2.

The code is provided in a Google Colab notebook: `https://colab.research.google.com/drive/1hBAKpO4Mg_jF5TEoVH24FFx2KRUsxTTk?usp=sharing`.