DATA SCIENCE – PART 2 | MIAS M1

Idriss JAIRI

Idriss.jairi@univ-lille.fr

COURSE PLAN

Session 1: Introduction to Data Science
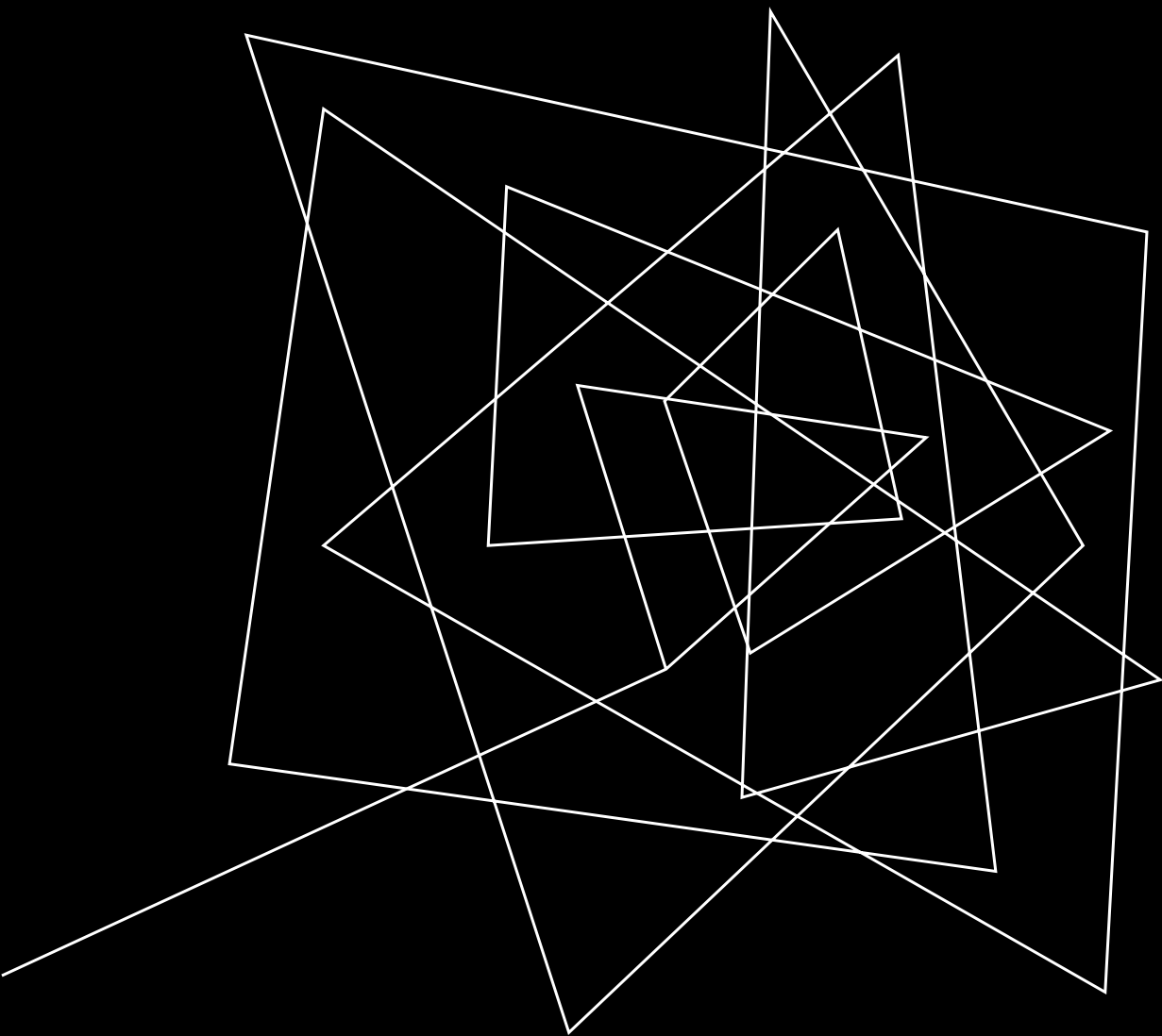
Session 2: Data Visualization and EDA

Session 3: Machine Learning Fundamentals

Session 4: Model Evaluation and Feature Engineering

Session 5: Introduction to Deep Learning

Session 6: Advanced Topics (Time series)

SESSION 1: INTRO TO DATA SCIENCE

Introduction to Data Science, Introduction to Python, Numerical Computing with NumPy and Data Manipulation with Pandas

Introduction to Data Science: What is Data Science?

Data science is the study of data to extract meaningful insights for business. It is a **multidisciplinary** approach that combines principles and practices from the fields of **mathematics**, **statistics**, **artificial intelligence**, and **computer engineering** to **analyze** large amounts of **data**. This analysis helps data scientists to ask and answer questions like **what happened**, **why it happened**, **what will happen**, and **what can be done with the results**.

This **multidisciplinary** field has the primary goal of identifying **trends**, **patterns**, **connections** and **correlations** in large data sets.

Introduction to Data Science: What is Data Science?

Using tools from **coding (computer science)**, **statistics and math**, and **domain knowledge** to work **creatively** and **efficiently** with **DATA.**

GOAL: Getting insights about the **DATA.**

Math and Statistics

Computer Science and IT

Domains/Business Knowledge

Introduction to Data Science: Why Data Science is Important?

Fig. Types of data.
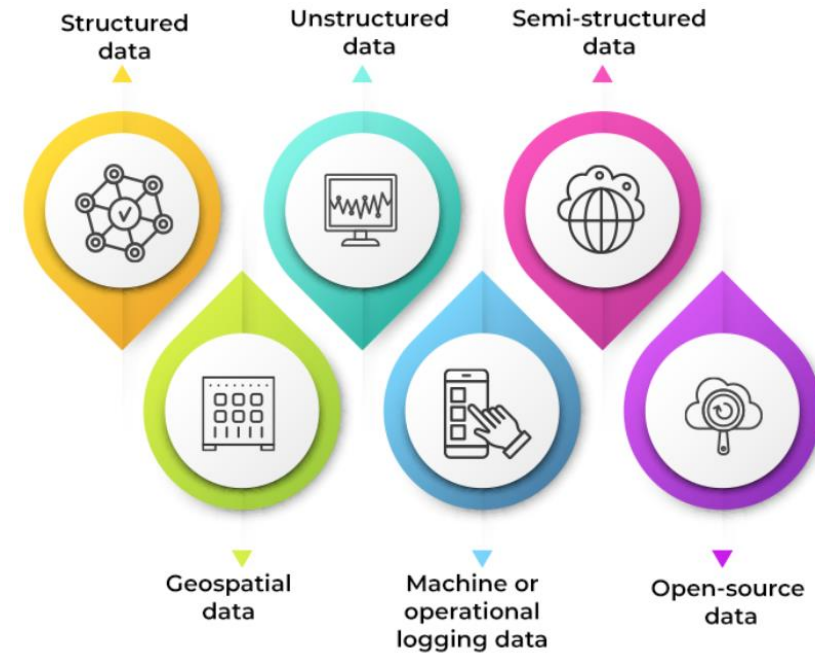**Source:** wikipedia.org

Fig. Types of big data.
Source: Spiceworks.com

Introduction to Data Science: Why Data Science is Important?

Data science is important because it combines tools, methods, and technology to generate meaning from data. Modern organizations are inundated with data; there is a proliferation of devices that can automatically collect and store information. Online systems and payment portals capture more data in the fields of e-commerce, medicine, finance, and every other aspect of human life. We have text, audio, video, and image data available in vast quantities.

Introduction to Data Science: Why Data Science is Important? <Some Key Reasons>

Informed Decision-Making

Data science enables organizations to make informed decisions by analyzing and interpreting large volumes of data. This leads to better strategies and improved decision outcomes.
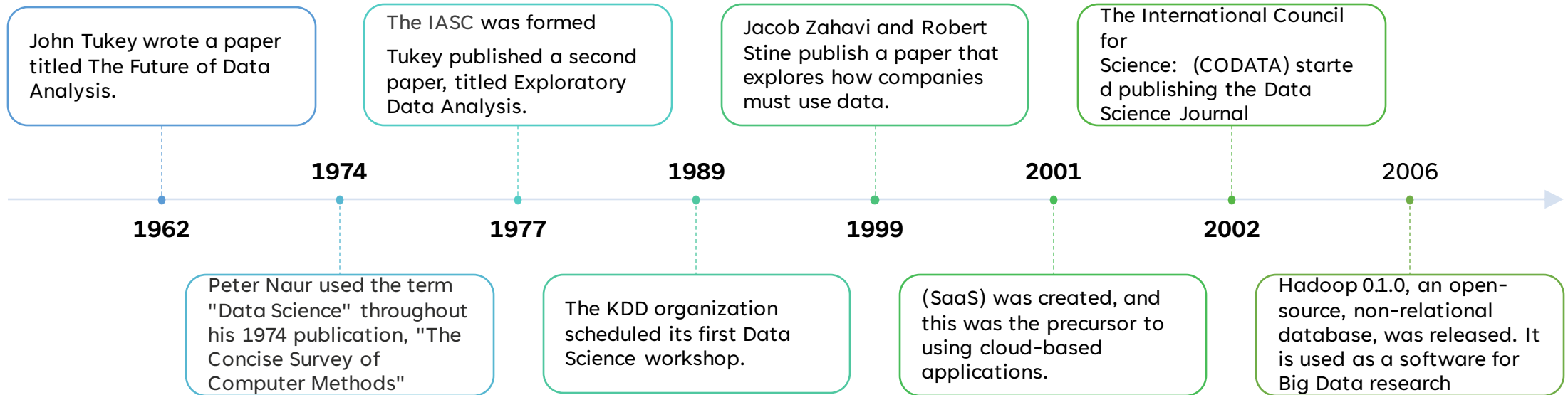
Business Intelligence

Data science helps businesses gain insights into customer behavior, market trends, and competitive landscapes. This intelligence is valuable for developing effective business strategies.

Predictive Analytics

Data science allows for the development of predictive models that can forecast future trends and outcomes. This is particularly beneficial in areas like sales forecasting, demand planning, and risk management.
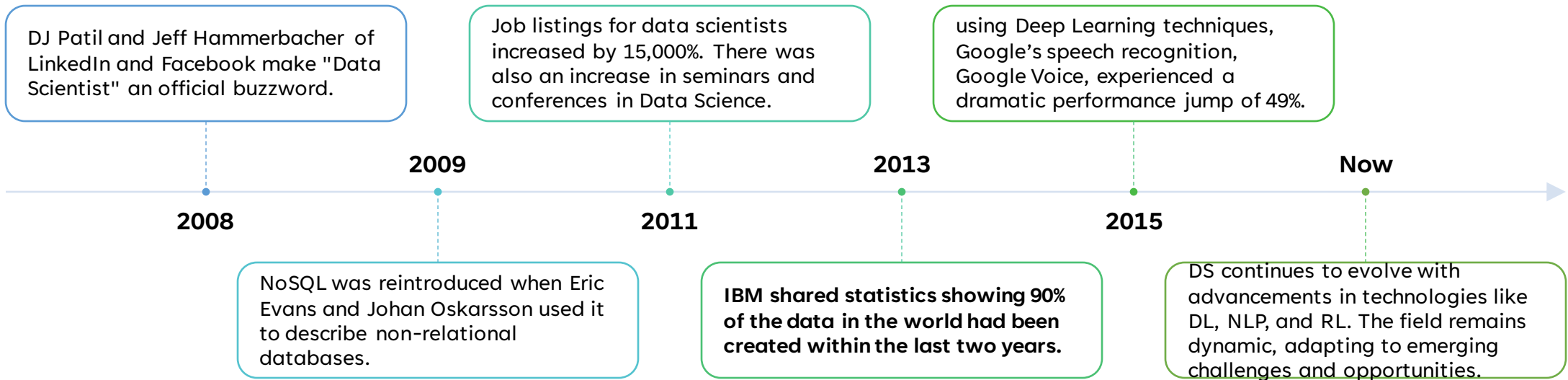
Introduction to Data Science: A Brief History

John Tukey wrote a paper titled The Future of Data Analysis.

The IASC was formed

Tukey published a second paper, titled Exploratory Data Analysis.

Jacob Zahavi and Robert Stine publish a paper that explores how companies must use data.

The International Council for Science: (CODATA) started publishing the Data Science Journal

1974

1989

2001

2006

1962

1977

1999

2002

Peter Naur used the term "Data Science" throughout his 1974 publication, "The Concise Survey of Computer Methods"

The KDD organization scheduled its first Data Science workshop.

(SaaS) was created, and this was the precursor to using cloud-based applications.

Hadoop 0.1.0, an open-source, non-relational database, was released. It is used as a software for Big Data research

Source 1: The History Of Data Science and Pioneers You Should Know
Source 2: A Brief History of Data Science
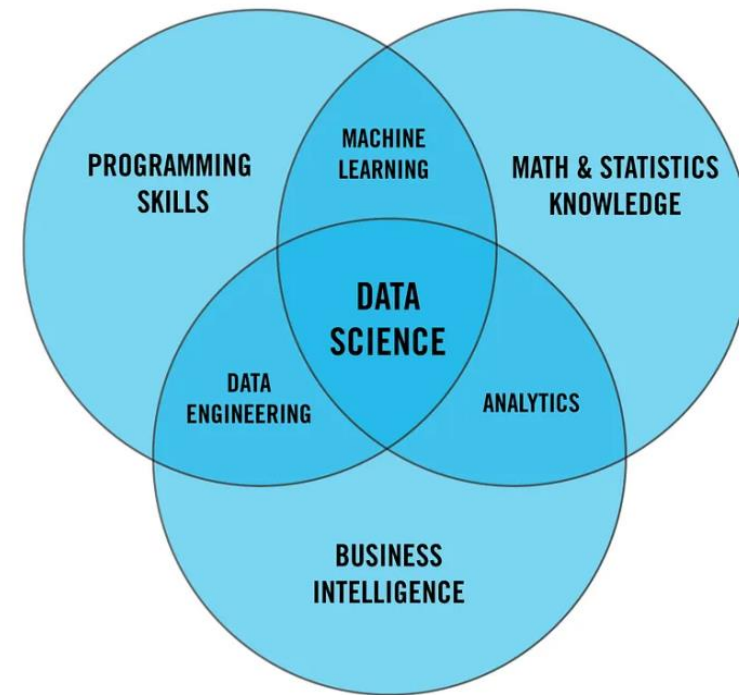
Introduction to Data Science: A Brief History

DJ Patil and Jeff Hammerbacher of LinkedIn and Facebook make "Data Scientist" an official buzzword.

Job listings for data scientists increased by 15,000%. There was also an increase in seminars and conferences in Data Science.

using Deep Learning techniques, Google's speech recognition, Google Voice, experienced a dramatic performance jump of 49%.

2009

2008

2011

2013

2015

Now

NoSQL was reintroduced when Eric Evans and Johan Oskarsson used it to describe non-relational databases.

IBM shared statistics showing 90% of the data in the world had been created within the last two years.

DS continues to evolve with advancements in technologies like DL, NLP, and RL. The field remains dynamic, adapting to emerging challenges and opportunities.

Source 1: The History Of Data Science and Pioneers You Should Know
Source 2: A Brief History of Data Science

10

Introduction to Data Science:
Data Scientist vs. Data Analyst vs. Data Engineer vs. Machine Learning Engineer?

The roles of **data scientists, data analysts, data engineers,** and **machine learning engineers** are distinct, yet they share some commonalities.

Introduction to Data Science:
Data Scientist vs. Data Analyst vs. Data Engineer vs. Machine Learning Engineer?

- **Data Scientist:**
  - **Focus:** Extracting insights and knowledge from data.
  - **Responsibilities:**
    - Analyzing and interpreting complex data sets.
    - Developing and implementing machine learning models.
    - Communicating findings to non-technical stakeholders.
    - Identifying trends, patterns, and correlations in data.
  - **Skills:**
    - Strong statistical and mathematical background.
    - Proficiency in programming languages (e.g., Python, R).
    - Data visualization and storytelling skills.
    - Machine learning expertise.

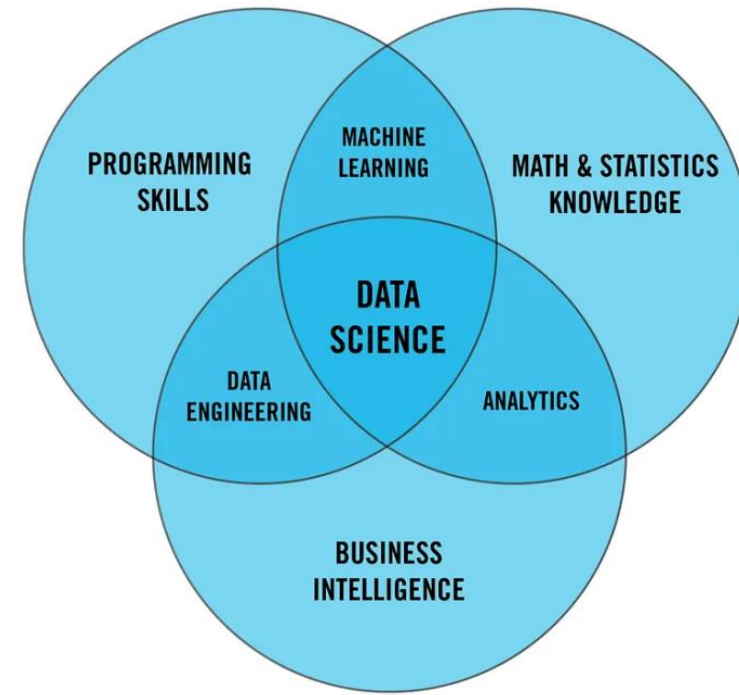# Introduction to Data Science:
Data Scientist vs. Data Analyst vs. Data Engineer vs. Machine Learning Engineer?

- **Data Analyst:**
  - **Focus:** Examining and interpreting data to provide insights.
  - **Responsibilities:**
    - Cleaning and processing data.
    - Creating reports and dashboards.
    - Identifying trends and patterns.
    - Conducting statistical analysis.
  - **Skills:**
    - Data cleaning and analysis using tools like SQL, Excel, or Python.
    - Data visualization skills.
    - Basic statistical knowledge.
    - Strong attention to detail.
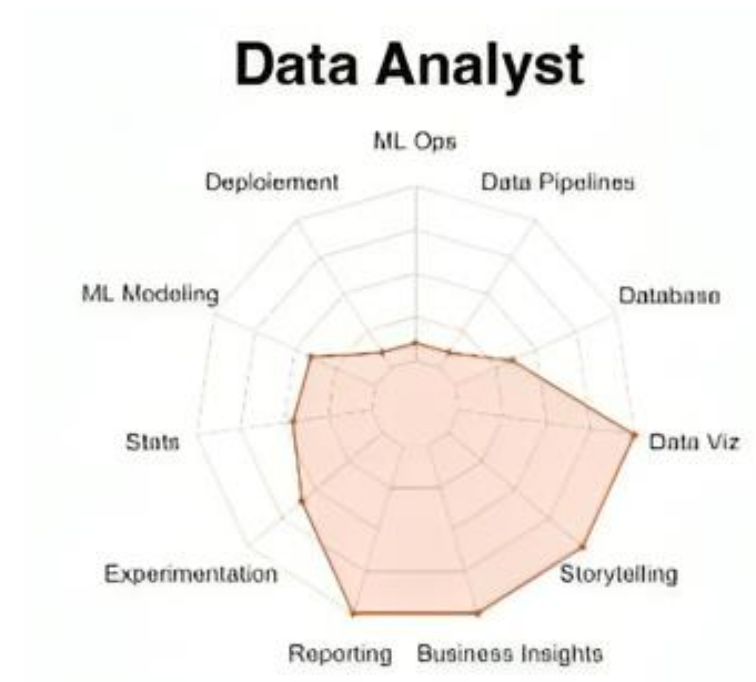
Introduction to Data Science:
Data Scientist vs. Data Analyst vs. Data Engineer vs. Machine Learning Engineer?

- **Data Engineer:**
 - **Focus:** Designing, constructing, and maintaining the systems and architecture for data generation.
  - **Responsibilities:**
    - Building and optimizing data pipelines.
    - Developing, constructing, testing, and maintaining databases.
    - Collaborating with data scientists to implement models into production.
  - **Skills:**
    - Proficiency in programming languages (e.g., Python, Java, Scala).
    - Knowledge of big data technologies (e.g., Hadoop, Spark).
    - Database design and management skills.
    - ETL (Extract, Transform, Load) processes.

Data Engineer

Introduction to Data Science:
Data Scientist vs. Data Analyst vs. Data Engineer vs. Machine Learning Engineer?

- **Machine Learning Engineer:**
  - **Focus:** Designing and implementing machine learning models into production systems.
  - **Responsibilities:**
    - Developing and deploying machine learning models.
    - Collaborating with data scientists and software engineers.
    - Ensuring scalability and performance of machine learning solutions.
  - **Skills:**
    - Strong programming skills (e.g., Python, Java).
    - In-depth knowledge of machine learning algorithms and frameworks.
    - Experience with model deployment and productionization.
    - Understanding of software engineering principles.

Introduction to Data Science:
Data Scientist vs. Data Analyst vs. Data Engineer vs. Machine Learning Engineer?

Note: While these roles have distinct responsibilities, in some organizations or smaller teams, individuals might wear multiple hats, and the lines between these roles can blur. Moreover, the field is dynamic, and the specific skills and responsibilities can vary based on the industry, company size, and evolving technologies.

Introduction to Data Science:
What Programming Languages Are Used in Data Science?

Data science involves a variety of programming languages, each serving different purposes in the data analysis workflow. The choice of programming language often depends on the specific task, preference, and the existing ecosystem.

The choice of programming language depends on factors such as the specific task, available libraries, ease of use, and the preferences of the data scientist or analyst. Many data scientists are proficient in multiple languages and choose the best tool for the job at hand.
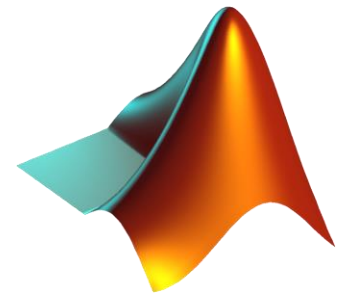
Introduction to Data Science:
What Programming Languages Are Used in Data Science?

Python: Python is widely used in the data science community due to its simplicity, readability, and a vast ecosystem of libraries and tools. Popular Python libraries for data science include NumPy, Pandas, Matplotlib, Seaborn, Scikit-learn, TensorFlow, and PyTorch.

R: R is a statistical programming language specifically designed for data analysis and visualization. It has a strong community in academia and is commonly used for statistical modeling, data exploration, and visualization.

SQL (Structured Query Language): SQL is essential for working with relational databases. It is used to retrieve, manipulate, and analyze data stored in databases. Knowledge of SQL is crucial for extracting and cleaning data from databases.

Introduction to Data Science: What Programming Languages Are Used in Data Science?

Julia: Julia is gaining popularity in the data science community for its performance and ease of use. It is designed to be fast and efficient for numerical and scientific computing, making it suitable for data-intensive tasks.

Scala: Scala, along with Apache Spark, is used for distributed data processing. Spark provides a fast and general-purpose cluster computing system, and Scala is the primary language for developing Spark applications.

Java: Java is commonly used in big data processing frameworks, such as Apache Hadoop. It's not as common in data science analysis but plays a significant role in handling large-scale data processing.

MATLAB: MATLAB is widely used in academia and industry for numerical computing and is popular for applications in engineering, physics, and finance.

centralelille

ufr3s
faculté
d'ingénierie et
management de la santé

Université
de Lille
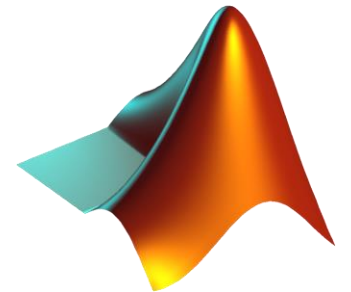
Introduction to Data Science:
What Software Are Used in Data Science?
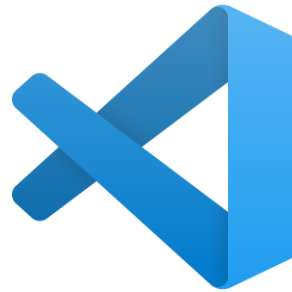
Introduction to Data Science: Data Science Workflow

The data science workflow refers to the series of steps or stages that data scientists follow to analyze and derive insights from data. While specific workflows may vary depending on the project, domain, or organization, a typical data science workflow often includes the following stages:

1. Define the Problem
2. Collect Data
3. Data Cleaning and Preprocessing
4. Exploratory Data Analysis (EDA)
5. Feature Engineering
6. Model Selection
7. Model Training
8. Model Evaluation
9. Interpretability and Explainability
10. Deployment
11. Monitoring and Maintenance
12. Documentation
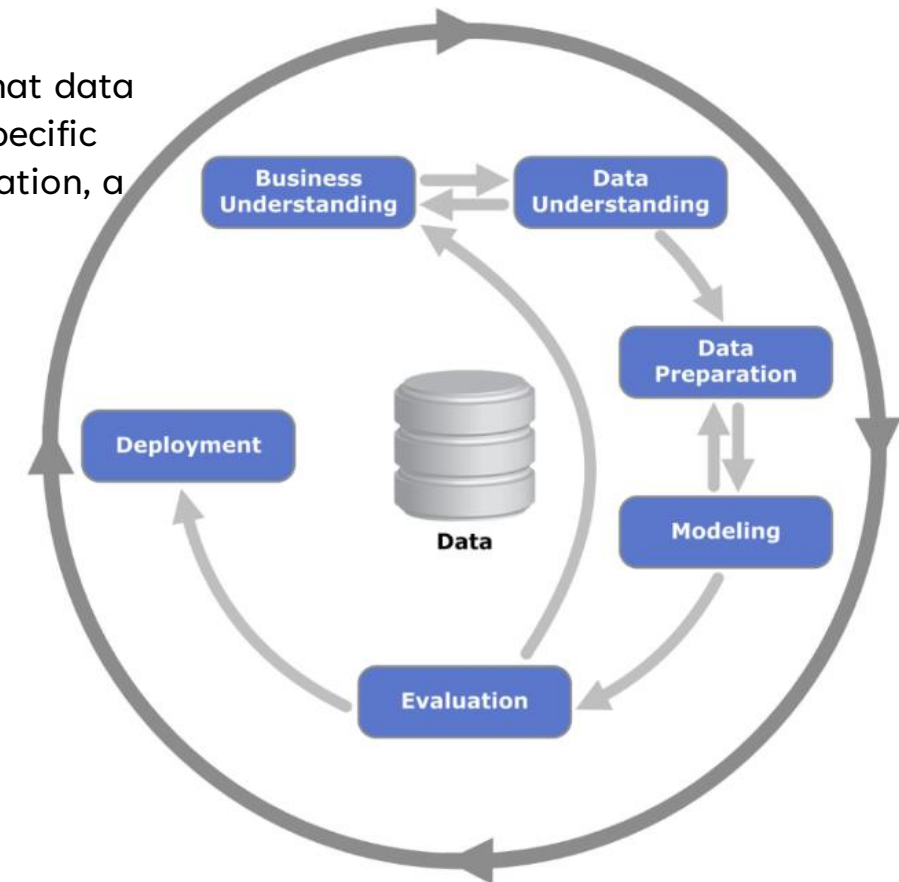
Introduction to Python: Overview

- **Brief History:**
  - Created by **Guido van Rossum** in the late 1980s
  - First released in 1991 (Python 1.0)
- **Key Features:**
  - Readability: Emphasizes code readability with a clean and easy-to-understand syntax.
  - Versatility: Supports both procedural and object-oriented programming paradigms.
  - Extensive Libraries: Offers a rich standard library and a vast ecosystem of third-party libraries.
  - Community Support: Active and supportive global community.
- **Applications:**
  - Web Development: Django, Flask.
  - Data Science: NumPy, Pandas, SciPy.
  - Artificial Intelligence: TensorFlow, PyTorch.
  - Automation: Scripting, task automation.

Guido van Rossum

Introduction to Python: Basic Syntax and Data Types

- Python uses indentation to define code blocks.
- Data Types:
  - Integers, Floats, Strings, Booleans.
  - Dynamic typing – no need to declare variable types explicitly.


```python
print("Hello, World!")
```

Introduction to Python: Variables, Data Structures, and Control Flow

- **Variables:**
  - **Declaration:** variable_name = value
  - **Example:** x = 10
- **Data Structures:**
  - **Lists:** Ordered, mutable, can contain different data types. Created using square brackets "[]"
  - **Tuples:** Ordered, immutable collections of elements. Created using parentheses "()"
  - **Dictionaries:** Unordered key-value pairs for efficient data retrieval. Created using curly braces "{}"
  - **Sets:** Unordered collections of unique elements. Created using curly braces "{}" or "set()" constructor

```python
my_list = [1, 2, "three", 4.0]
```

```python
my_tuple = (1, 2, "three", 4.0)
```

```python
my_dict = {"name": "John", "age": 25}
```

```python
my_set = {1, 2, 3, 4, 5}
```

Introduction to Python: Variables, Data Structures, and Control Flow
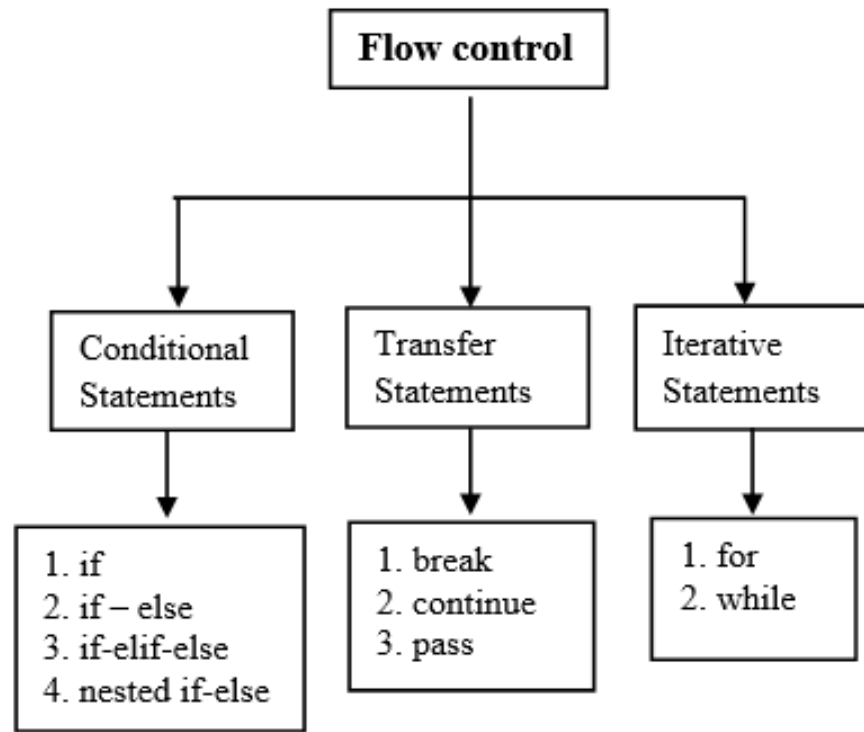

```
if condition:
    # code block
else:
    # code block
```

```
for item in my_list:
    # code block
```

```
while condition:
    # code block
```

Introduction to Python: Learning Resources

Link: https://www.python.org/

Link: https://www.w3schools.com/python/
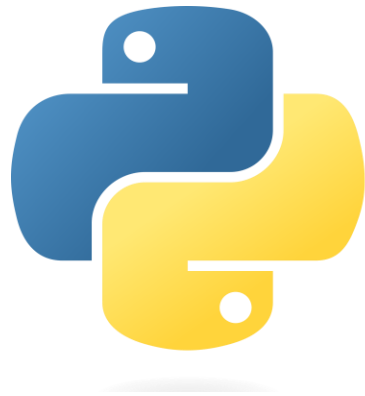
Introduction to Python: DEMO

DEMO: [Session 1 – Intro to Data Science]

Numerical Computation with NumPy: <Definition>

- **Definition:** NumPy, short for Numerical Python, is a fundamental library for numerical computing in Python. Its core feature is the ndarray object, which allows efficient manipulation of large, multi-dimensional arrays and matrices. NumPy plays a pivotal role in scientific computing, data analysis, and machine learning, making it an indispensable tool for researchers, engineers, and data scientists.
- **Why choose NumPy for numerical computing?** NumPy excels in handling large datasets efficiently, providing optimized mathematical operations on arrays, and offering a broad range of mathematical functions. Its ability to seamlessly integrate with other libraries, such as SciPy for scientific computing and Matplotlib for data visualization, makes NumPy a cornerstone in the Python data science ecosystem."

N-dimensional Arrays

NumPy's main feature is the numpy.ndarray data structure, which is a multi-dimensional array. It allows you to represent and manipulate arrays of any dimensionality efficiently.
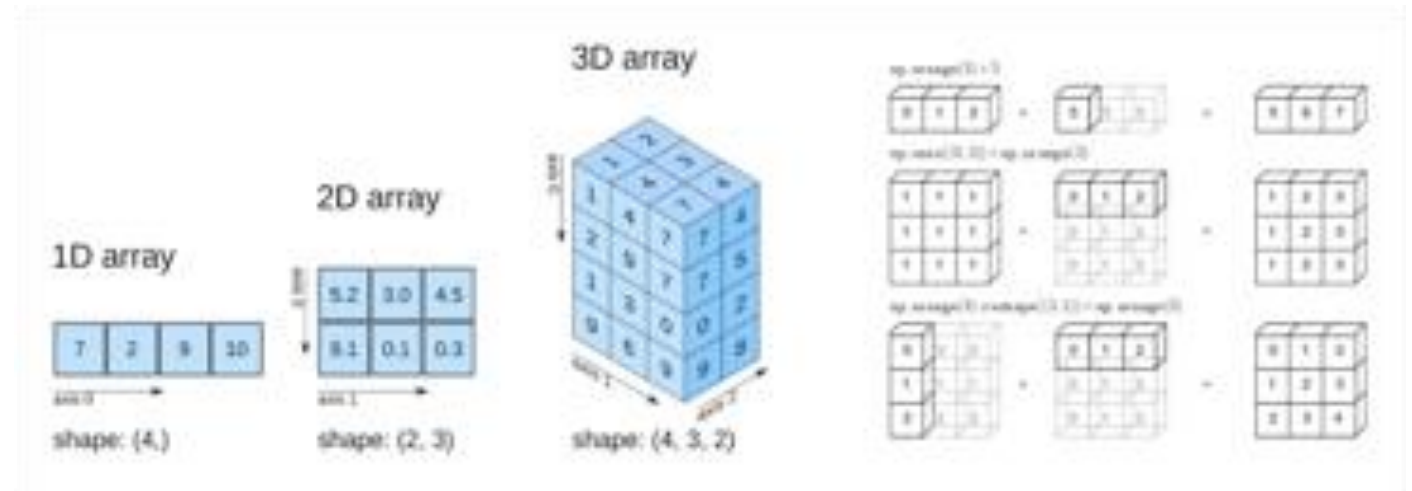
Broadcasting

NumPy supports broadcasting, a powerful mechanism that allows operations between arrays of different shapes and sizes. Broadcasting automatically expands smaller arrays to match the shape of larger ones, making element-wise operations more flexible.

multiplying several columns at once

row-wise normalization

outer product

Numerical Computation with NumPy: <Features>

Universal Functions (ufuncs)

- NumPy provides a set of universal functions that operate element-wise on arrays. These include mathematical, trigonometric, bitwise, and other operations. Ufuncs allow for efficient array computations without the need for explicit loops.

- **Why use ufuncs?** ufuncs are used to implement vectorization in NumPy which is way faster than iterating over elements. They also provide broadcasting and additional methods like reduce, accumulate etc. that are very helpful for computation.

```python
import numpy as np

x = [1, 2, 3, 4]
y = [4, 5, 6, 7]

z = np.add(x, y)

print(z)
```
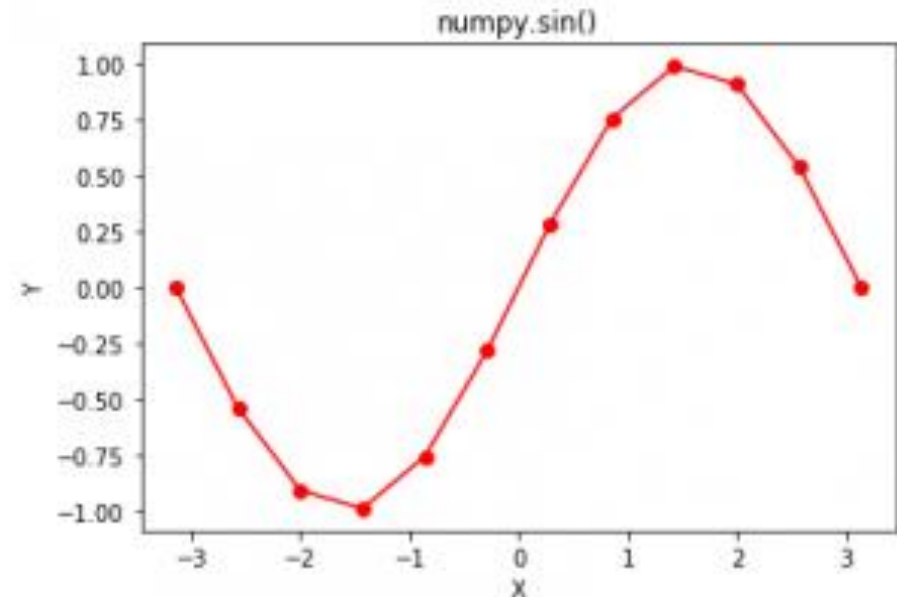
Numerical Computation with NumPy: <Features>

Mathematical Functions

NumPy includes a wide range of mathematical functions for linear algebra, Fourier analysis, random number generation, statistical analysis, and more. Examples include np.sin(), np.cos(), np.exp(), np.mean(), and many others.

Indexing Arrays

Array indexing is the same as accessing an array element.

- You can access an array element by referring to its index number.
- The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.
- **Note:** Use negative indexing to access an array from the end.


```
>>> a[0,3:5]
array( [3,4] )

>>> a[4:, 4:]
array( [ 28, 29],
       [ 34, 35] ] )

>>> a[ :, 2]
array( [2, 8, 14, 20, 26, 32] )

>>> a[2 : : 2, : : 2]
array( [ 12, 14, 16],
       [ 24, 26, 28] ] )
```

Numerical Computation with NumPy:
<Features>

Indexing Arrays


```python
import numpy as np


arr_1d = np.array([1, 2, 3, 4, 5])


# Accessing a single element
print(arr_1d[2])   # Output: 3


# Negative indexing
print(arr_1d[-1])   # Output: 5
```



```python
arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])


# Accessing a single element
print(arr_2d[1, 2])   # Output: 6


# Accessing an entire row
print(arr_2d[1])   # Output: [4 5 6]


# Accessing an entire column
print(arr_2d[:, 1])   # Output: [2 5 8]
```
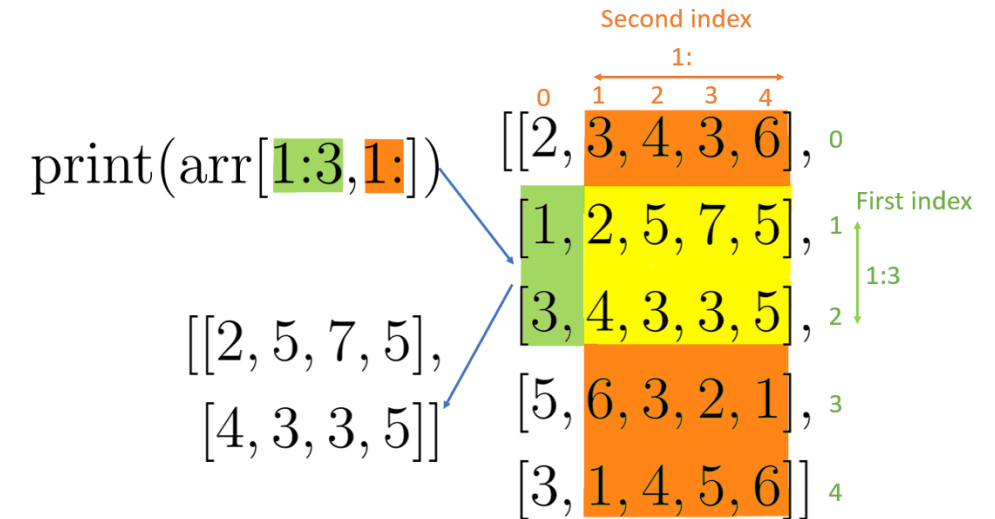
1D Array   **2D Array**

Slicing Arrays

Slicing in python means taking elements from one given index to another given index.

- o We pass slice instead of index like this: [start:end].
- o We can also define the step, like this: [start:end:step].
- o If we don't pass start it's considered 0
- o If we don't pass end it's considered length of array in that dimension
- o If we don't pass step it's considered 1
- o **Note:** The result includes the start index, but excludes the end index.

$$\text{print}(\text{arr}[1:3,1:])$$

$$[[2, 5, 7, 5],$$
$$[4, 3, 3, 5]]$$

Numerical Computation with NumPy: <Features>

Slicing Arrays

```
arr_1d = np.array([1, 2, 3, 4, 5])

# Slicing a subarray
subarray = arr_1d[1:4]  # Output: [2 3 4]


# Slicing with a step
subarray_step = arr_1d[::2]  # Output: [1 3 5]
```

1D Arrays

```
arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Slicing a subarray
subarray = arr_2d[0:2, 1:3]
# Output:
# [[2 3]
#  [5 6]]

# Slicing with a step
subarray_step = arr_2d[:, ::2]
# Output:
# [[1 3]
#  [4 6]
#  [7 9]]
```

2D Arrays

Numerical Computation with NumPy: <Features>

Linear Algebra Operations:

NumPy provides a comprehensive set of linear algebra operations, including matrix multiplication (np.dot()), determinant calculation (np.linalg.det()), eigenvalue computation (np.linalg.eig()), and more.

```python
import numpy as np

# Define two matrices
matrix1 = np.array([[1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9]])

matrix2 = np.array([[9, 8, 7],
                    [6, 5, 4],
                    [3, 2, 1]])

# Perform matrix multiplication
result_matrix = np.dot(matrix1, matrix2)
```

Numerical Computation with NumPy:
<NumPy Installation and Importing>

Anaconda Prompt (anaconda3)

```
(base) C:\Users\jairiidriss>pip install numpy
```
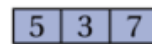
```
In [50]: import numpy as np
```
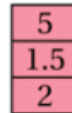
Numerical Computation with NumPy:
<Review of Linear Algebra>

- **Vectors and vector operations**
- **Matrix and matrix operations**

Source: https://web.stanford.edu/class/cs246/
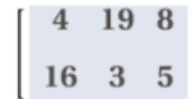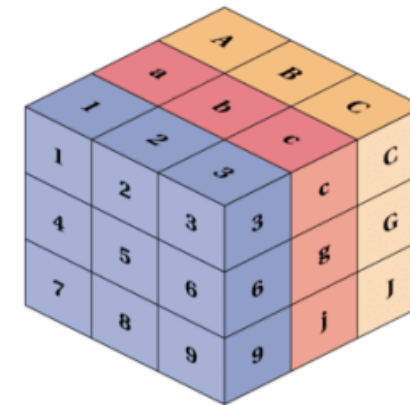
Numerical Computation with NumPy: <Review of Linear Algebra>

Why Linear Algebra?

Link: Introduction to Linear Algebra for Applied Machine Learning with Python

Numerical Computation with NumPy:
<Review of Linear Algebra>

Vectors and vector operations: Column vs. Row vector

A vector is a mathematical object that has both magnitude and direction. It is often represented as an ordered list of numbers or coordinates.

$$V = \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{bmatrix} (n * 1)$$

$$V = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix} : (1 * n)$$

Row vector

Column vector

Numerical Computation with NumPy: <Review of Linear Algebra>

Vectors and vector operations: Vector Multiplication (dot product)

$$u \cdot v = \begin{bmatrix} u_1 \\ u_2 \\ u_n \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_n \end{bmatrix} = u_1.v_1 + u_2.v_2 + ... + u_n.v_n = \sum_{i=1}^{n} u_i.v_i$$

Numerical Computation with NumPy: <Review of Linear Algebra>

Vectors and vector operations: Norm of a vector

The norm of a vector is a mathematical concept that represents the size or length of the vector. There are different ways to calculate the norm, and the choice of norm depends on the context and application. The two most common norms are the Euclidean norm (or 2-norm) and the Manhattan norm (or 1-norm).

- Euclidean Norm (L2 Norm): The Euclidean norm of a vector v in n-dimensional space is denoted as $\|v\|$ or $\|v\|_2$
- Manhattan Norm (L1 Norm): The Manhattan norm of a vector v is denoted as $\|v\|_1$, and it is calculated as the sum of the absolute values of its components

$$\|\mathbf{v}\| = \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2}$$

$$\|\mathbf{v}\|_1 = |v_1| + |v_2| + \ldots + |v_n|$$
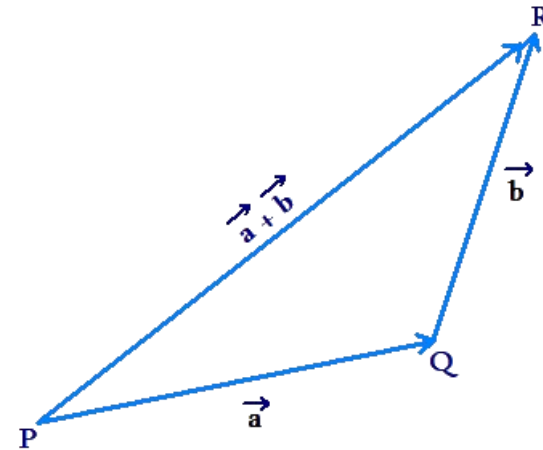
Numerical Computation with NumPy:
<Review of Linear Algebra>

Vectors and vector operations: Triangle Inequality

For two vectors u and v, we have:

$$||u + v|| \leq ||u|| + ||v||$$

$$||u - v|| \geq \Big| ||u|| - ||v|| \Big|$$

Numerical Computation with NumPy:
<Review of Linear Algebra>

Matrix and Matrix Operations

- A matrix is a rectangular array of numbers, symbols, or expressions, arranged in rows and columns. The size of a matrix is given by its number of rows and columns.
- Matrices are denoted by uppercase letters. For example, an m×n matrix A could be represented as:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Numerical Computation with NumPy: <Review of Linear Algebra>

Matrix Operations: Addition

Matrix addition is defined for matrices of the same dimension. Matrices are added component-wise:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix}$$

Numerical Computation with NumPy:
<Review of Linear Algebra>

Matrix Operations: Multiplication

Matrices can be multiplied like so:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 \times 5 + 2 \times 7 & 1 \times 6 + 2 \times 8 \\ 3 \times 5 + 4 \times 7 & 3 \times 6 + 4 \times 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

You can also multiply non-square matrices, but the dimensions have to match(i.e. the number of columns of the first matrix has to equal the number of rows of the second matrix).

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 2 \cdot 4 & 1 \cdot 2 + 2 \cdot 5 & 1 \cdot 3 + 2 \cdot 6 \\ 3 \cdot 1 + 4 \cdot 4 & 3 \cdot 2 + 4 \cdot 5 & 3 \cdot 3 + 4 \cdot 6 \\ 5 \cdot 1 + 6 \cdot 4 & 5 \cdot 2 + 6 \cdot 5 & 5 \cdot 3 + 6 \cdot 6 \end{bmatrix} = \begin{bmatrix} 9 & 12 & 15 \\ 19 & 26 & 33 \\ 29 & 40 & 51 \end{bmatrix}$$

In general, If matrix A is multiplied by matrix B, we have $(AB)_{ij} = \Sigma_k A_{i,k} * B_{k,j}$ for all entries (i,j)of the matrix product.

$$(AB)_{ij} = \sum_k A_{ik} B_{kj}$$

Matrix Operations: Multiplication

Note: Matrix multiplication is **associative**, i.e. (AB)C=A(BC). It is also **distributive**, i.e. A(B+C)=AB+AC. However, it is **not commutative**. That is, AB does not have to equal BA. Note that if you multiply a 1-by-n matrix with an n-by-1 matrix, that is the same as taking the dot product of the corresponding vectors.

Matrix Operations: Matrix Transpose

The transpose operation switches a matrix's rows with its columns, so:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^{T} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

In other words, we define A^T by $(A^T)_{ij} = A_{ji}$.

Properties:
- ○ $(A^T)^T = A$
- ○ $(AB)^T = B^T A^T$
- ○ $(A+B)^T = A^T + B^T$

Matrix Operations: Identity Matrix

The identity matrix I_n is an n-by-n matrix with all 1's on the diagonal, and 0's everywhere else. It is usually abbreviated I, when it is clear what the dimensions of the matrix are. It has the property that when you multiply it by any other matrix, you get that matrix. In other words, if A is an m-by-n matrix, then $A*I_n = I_m*A = A$.

$$I_1 = [1], \; I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \; I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; \dots, \; I_n = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}.$$
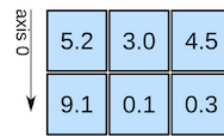
Numerical Computation with NumPy: <DEMO>

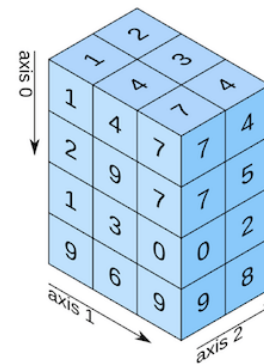**Demo:** [Session 1 – Introduction to Data Science](#)
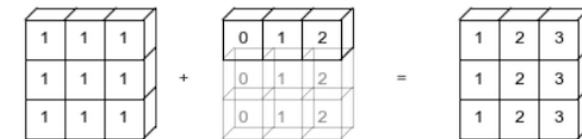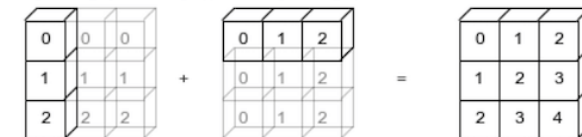
Data Manipulation With Pandas: Introduction

- **Definition 1: Pandas** is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.
- **Definition 2: Pandas** stands as a cornerstone in the realm of data manipulation within the Python ecosystem. It is a robust and versatile library designed to facilitate the handling and analysis of structured data seamlessly.
- **Programming languages:** Python, C, Cython
- **Developer:** Wes McKinney
- **Initial release:** 11 January 2008; 15 years ago

Data Manipulation With Pandas: Brief History and Purpose of Pandas

The story of **Pandas** begins with its creation by **Wes McKinney** in **2008** while working in the financial industry. Faced with the need for an efficient tool to manipulate and analyze financial data, McKinney developed Pandas, which was later open-sourced in 2009. Since then, it has evolved into one of the most essential libraries for data manipulation and analysis in Python.
The primary purpose of Pandas is to provide easy-to-use data structures and functions, enabling users to manipulate, analyze, and visualize structured data seamlessly. It caters to a diverse range of applications, from cleaning and preprocessing datasets to exploratory data analysis and statistical modeling.

Wes McKinney

Data Manipulation With Pandas: Key Features <DataFrame and Series>

At the core of Pandas lie two fundamental data structures: the **DataFrame** and the **Series**.

- The **DataFrame** is a two-dimensional, tabular data structure resembling a spreadsheet, where data is organized into rows and columns. It allows for efficient manipulation of structured data and is well-suited for tasks ranging from data cleaning to complex analytics.
- The Series is a one-dimensional array-like object that can hold any data type. It serves as the building block for constructing DataFrames and offers powerful indexing and data manipulation capabilities.

Understanding these key features is pivotal for unleashing the full potential of Pandas in your data manipulation endeavors.

Data Manipulation With Pandas: Loading and Exploring Datasets

- Importing Pandas and Common Aliases
    - import pandas as pd
- Loading Data from Various Sources
    - Using "pd.read_*" function
        - pd.read_csv() for CSV files.
        - pd.read_excel() for Excel spreadsheets.
        - pd.read_sql() for SQL databases.

```python
import pandas as pd
```

```
import pandas as pd

# Read a CSV file into a DataFrame
df = pd.read_csv('your_file.csv')

# Display the DataFrame
print(df)
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Basic Exploration Methods>

Basic Exploration Methods: **head**(), **tail**(), **info**(), **describe**().
Pandas offers a suite of quick and powerful methods to gain a snapshot of your dataset:

- **head**(): Displays the first few rows of your DataFrame, allowing you to glimpse at the data's initial structure.
- **tail**(): Provides a view of the last few rows, aiding in understanding the tail-end of your dataset.
- **info**(): Offers a concise summary of the DataFrame, including the data types, non-null counts, and memory usage.
- **describe**(): Generates descriptive statistics, giving a statistical overview of the dataset's numeric columns.

```
# Display the first 5 rows
print(df.head())
```

```
# Display the last 5 rows
print(df.tail())
```

```
# Display DataFrame information
print(df.info())
```

```
# Display summary statistics
print(df.describe())
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Accessing Columns and Rows>

```python
# Access a specific column by name
column_data = df['column_name']
```

- **Accessing Columns:**
  - By Column Name
  - Multiple Columns
- **Accessing Rows:**
  - By Index (loc)
  - By Integer Position (iloc)
  - Slicing Rows

```python
# Access multiple columns
selected_columns = df[['column1', 'column2']]
```

```python
# Access a specific row by index label
row_data = df.loc[row_index]
```

```python
# Access a specific row by integer position
row_data = df.iloc[row_position]
```

```python
# Access a range of rows using slicing
sliced_data = df[2:5]  # Rows 2 to 4
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Accessing Columns and Rows>

Accessing Both Rows and Columns
- o By Index and Column Name (Ioc)
- o By Integer Position and Column Position (iloc)

```
# Access a specific element by row index and column name
element_data = df.loc[row_index, 'column_name']
```

```
# Access a specific element by integer position and column position
element_data = df.iloc[row_position, column_position]
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Checking for Missing Values>

```python
# Check for missing values in the entire DataFrame
print(df.isnull().sum())
```

```python
# Check for missing values in the entire DataFrame
print(df.isna().sum())
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Basic Data Cleaning>

Identifying and Handling Missing Data: **dropna**() and **fillna**():

- o **'dropna**()': This method allows us to remove rows or columns containing missing values. By specifying the axis, we can choose to drop entire rows or columns.
- o **'fillna**()': Alternatively, we can fill in missing values with specific values or strategies. This method is particularly useful when outright removal of missing data is not desirable.

```
# Drop rows with any missing values
df_cleaned = df.dropna()
```

```
# Drop columns with any missing values
df_cleaned = df.dropna(axis=1)
```

```
# Fill missing values with a specific value (e.g., 0)
df_filled = df.fillna(0)
```

```
# Fill missing values with the mean of the column
df_filled = df.fillna(df.mean())
```

Removing Duplicates: **drop_duplicates**()
Duplicate entries can distort analyses and lead to misleading results. Pandas provides a simple solution to eliminate these redundancies.

- **drop_duplicates**(): This method removes duplicate rows, retaining only the first occurrence.

```python
# Remove duplicate rows based on all columns
df_no_duplicates = df.drop_duplicates()
```

```python
# Remove duplicates based on specific columns
df_no_duplicates = df.drop_duplicates(subset=['column1', 'column2'])
```

Renaming Columns for Clarity

Column names play a vital role in understanding and communicating the content of a dataset. Pandas facilitates easy renaming of columns.

- o   rename(): Use this method to rename one or more columns, providing a clear and concise representation of the data.

```python
# Rename a single column
df.rename(columns={'old_name': 'new_name'}, inplace=True)
```

```python
# Rename multiple columns
df.rename(columns={'old_name1': 'new_name1', 'old_name2': 'new_name2'}
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Basic Data Manipulation>

- Filtering Data with Boolean Indexing, Boolean indexing is a powerful method for selectively extracting data from a DataFrame based on conditions.
- Filter data with boolean indexing using 'and' (&) and 'or' (|) conditions.

```python
# Filtering data based on a condition
filtered_data = df[df['column_name'] > threshold_value]
```

```python
# Filtering data with 'and' condition
filtered_and_data = df[(df['Age'] > 25) & (df['Salary'] > 55000)]
```

```python
# Filtering data with 'or' condition
filtered_or_data = df[(df['Department'] == 'IT') | (df['Salary'] > 60000)]
```

Selecting Specific Columns and Rows using **loc** and **iloc**: Pandas offers two primary methods for precise selection of data: **loc** and **iloc**.
- o **'loc'**: This method allows selection by label. We can specify both the rows and columns we want to extract.
- o **'iloc'**: This method facilitates integer-location based indexing, providing a powerful tool for numeric selection.

```python
# Select specific rows and columns by label
selected_data = df.loc[row_labels, ['column1', 'column2']]
```

```python
# Select specific rows and columns by integer position
selected_data = df.iloc[row_positions, colum_positions]
```

Adding and removing columns:
- o   Adding columns
- o   Removing columns

```python
# Add a new column with a calculated value
df['new_column'] = df['column1'] + df['column2']
```

```python
# Remove a column
df.drop(['column_to_remove'], axis=1, inplace=True)
```

Applying Functions to Columns using apply()

The apply() function in Pandas is a powerful tool for transforming data within columns. It allows the application of custom or predefined functions to each element or column of a DataFrame.

```python
# Applying a function to a column
df['new_column'] = df['existing_column'].apply(lambda x: custom_function(x))
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Basic Data Manipulation>

Grouping and Aggregating Data with groupby()

- o Grouping and aggregation are essential steps in data analysis, enabling us to draw meaningful conclusions from our datasets.
- o The groupby() function allows us to group data based on specific criteria and perform aggregate operations, such as calculating means, counts, or custom functions, on each group.

```python
# Grouping data by a column and calculating aggregate statistics
grouped_data = df.groupby('grouping_column').agg({'numeric_column': 'mean', 'other_column': 'count'})
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Basic Data Manipulation>

Merging and **Concatenating** DataFrames: Combining data from multiple sources is a common requirement in real-world data analysis. Pandas provides methods for both merging and concatenating DataFrames.

```python
# Merge two DataFrames based on a common column
merged_data = pd.merge(df1, df2, on='common_column', how='inner')
```

```python
# Concatenate DataFrames along a specific axis (rows or columns)
concatenated_data = pd.concat([df1, df2], axis=0)
```

Introduction to Pivot Tables

Pivot tables are a powerful tool for reshaping and summarizing data, providing a concise and structured view of information.
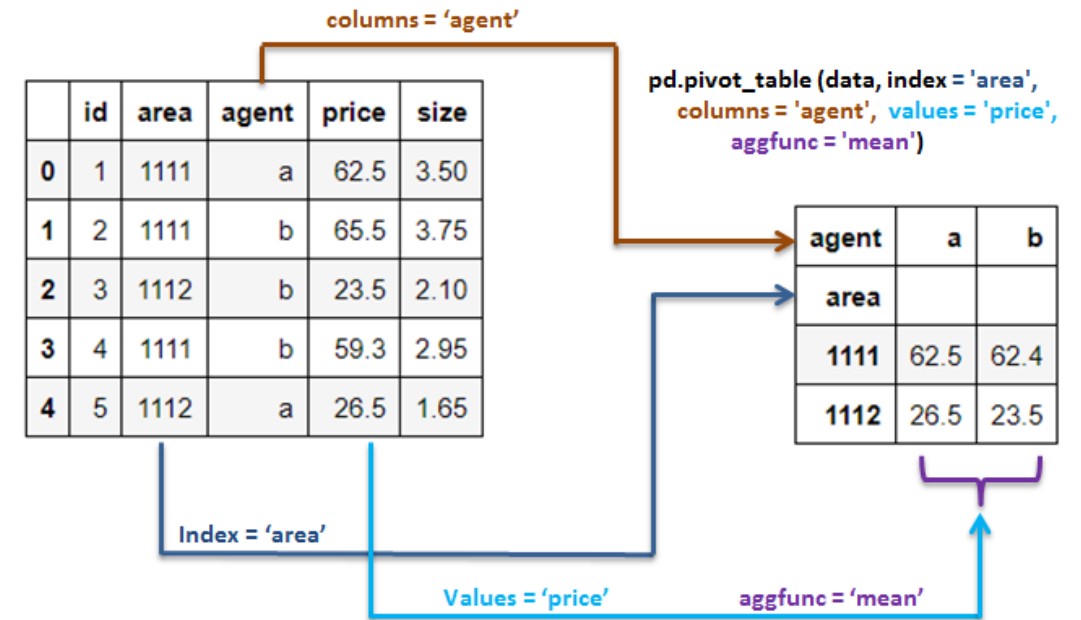
Data Manipulation With Pandas: Learning Resources

Link: https://pandas.pydata.org/

Link: https://www.w3schools.com/python/pandas/

Data Manipulation with Pandas: <DEMO>

Demo: Session 1 – Introduction to Data Science

Course's Practical Example: Titanic Disaster

Demo: Session 1 – Introduction to Data Science