

```
from IPython.display import Image, display  
  
# Display the image  
display(Image(filename='Lecture 1.png'))
```



What is data?

- Raw facts and figures
- Can be numbers, text, images, or signals
- Basis for analysis and decision-making

Why is data important today?

- Drives evidence-based decisions
- Enables automation and AI
- Helps organizations gain insights and competitive advantage

Real-life examples:

- Healthcare: patient records, medical imaging, wearable devices
- Business: sales transactions, customer behavior
- Climate: temperature, CO₂ levels, satellite data
- Social Media: posts, likes, shares, user interactions

Types of Data Based on Structure

1. Structured Data

Data organized in a predefined format (rows, columns, schemas).

Storage: Relational databases (SQL), spreadsheets.

Examples:

Patient records in a hospital database (age, gender, diagnosis).

Financial transactions (date, amount, account number).

2. Semi-Structured Data

Data with some structure but not rigid like relational databases. It may use tags, keys, or hierarchies.

Storage: JSON, XML, NoSQL databases.

Examples: JSON logs from IoT devices.

Web data in XML or HTML.

3. Unstructured (Non-Structural) Data

Data without a predefined format, messy and harder to analyze directly.

Storage: Data lakes, distributed storage systems.

Examples:

Text: Medical notes, research articles, social media posts.

Images: X-rays, MRI scans, cow images for milk prediction.

Audio: Doctor-patient conversations, voice assistants.

Video: Surgery recordings, CCTV footage.

Types of Data Based on Categories & Numeric

- Categories: These represent labels, names, or categories rather than numbers.
- Numeric: These represent numbers and can be measured.

Types of categories data

- Nominal → Categories with no order.

Examples: Gender (Male/Female), Blood type (A, B, AB, O), Country names.

- Ordinal → Categories with a meaningful order, but differences aren't measurable.

Examples: Education level (High School < Bachelor < Master < PhD), Customer satisfaction (Poor, Fair, Good, Excellent)

Types of numeric data

- Discrete → Countable numbers (whole numbers).

Examples: Number of patients, number of cows on a farm, Test scores.

- Continuous → Measured values that can take any value within a range.

Examples: Height, Weight, Temperature, Milk yield in liters.

Raw Data Challenges

1. Incomplete Values

Missing entries in datasets

Example: blank cells in a survey

2. Duplicates

Same records repeated

Example: duplicate customer entries

3. Noise & Errors

Incorrect or inconsistent data

Example: typo in age ("250 years old")

4. Different Formats

Data stored in various structures

Example: dates as DD/MM/YYYY vs. MM-DD-YYYY

How to install the environment for writing codes?

- Install Anaconda <https://www.anaconda.com/download/success>
- Installing Jupyter
- Setup Environment

The main libraries and how they are installed

- Pandas → data manipulation and analysis

pip install pandas

- NumPy → numerical operations, arrays, linear algebra

pip install numpy

- scikit-learn → machine learning algorithms

pip install scikit-learn

- TensorFlow → deep learning framework

pip install tensorflow

- Keras → high-level neural networks API (on top of TensorFlow)

pip install keras

- XGBoost gradient boosting libraries

pip install xgboost

- Matplotlib → 2D plotting

pip install matplotlib

- Seaborn → statistical data visualization

pip install seaborn

What is variable

A variable is something that can hold a value, and that value may change over time.

```
age = 25    # variable 'age' stores the value 25
name = "Hager" # variable 'name' stores a string
age=400
```

For loop

```
# Example with a list
numbers = [10, 20, 30, 40]

for n in numbers:
    print(n)

10
20
30
40
```

```

# Iterating over dictionaries
student = {"name": "Hager", "age": 25, "country": "Ireland"}

for key, value in student.items():
    print(key, ":", value)

name : Hager
age : 25
country : Ireland

# range() in for loop
for i in range(4, 5):
    print("Index:", i)

Index: 0
Index: 1
Index: 2
Index: 3
Index: 4

```

if condition

```

# if condition
age = 18

if age >= 18:
    print("You are an adult.")

You are an adult.

# if ... else
age = 16

if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")

You are a minor.

# if ... elif ... else
score = 75

if score >= 90:
    print("Excellent")
elif score >= 60:
    print("Pass")
else:
    print("Fail")

```

Pass

DataFrame

What is a DataFrame?

A DataFrame is a 2-dimensional, table-like data structure in pandas (a Python library for data analysis). It looks like an Excel spreadsheet or a SQL table: it has rows and columns with labels.

Rows → represent records/observations (indexed by numbers or labels).

Columns → represent variables/features (each column has a name).

```
import pandas as pd

data = {
    "Name": ["Alice", "Bob", "Hager"],
    "Age": [25, 30, 28],
    "Country": ["USA", "UK", "Ireland"]
}
```

```
df = pd.DataFrame(data)
print(df)
```

	Name	Age	Country
0	Alice	25	USA
1	Bob	30	UK
2	Hager	28	Ireland

Downloading and Reading Dataset

Loading dataset

```
df = pd.read_csv('Diabetes Missing Data.csv')
df.head()
```

	Pregnant	Glucose	Diastolic_BP	Skin_Fold	Serum_Insulin	BMI	\
0	6	148.0	72.0	35.0	NaN	33.6	
1	1	85.0	66.0	29.0	NaN	26.6	
2	8	183.0	64.0	NaN	NaN	23.3	
3	1	89.0	66.0	23.0	94.0	28.1	
4	0	137.0	40.0	35.0	168.0	43.1	

	Diabetes_Pedigree	Age	Class
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1

```
df.head(10)
```

	Pregnant	Glucose	Diastolic_BP	Skin_Fold	Serum_Insulin	BMI \
0	6	148.0	72.0	35.0	NaN	33.6
1	1	85.0	66.0	29.0	NaN	26.6
2	8	183.0	64.0	NaN	NaN	23.3
3	1	89.0	66.0	23.0	94.0	28.1
4	0	137.0	40.0	35.0	168.0	43.1
5	5	116.0	74.0	NaN	NaN	25.6
6	3	78.0	50.0	32.0	88.0	31.0
7	10	115.0	NaN	NaN	NaN	35.3
8	2	197.0	70.0	45.0	543.0	30.5
9	8	125.0	96.0	NaN	NaN	NaN

	Diabetes_Pedigree	Age	Class
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1
5	0.201	30	0
6	0.248	26	1
7	0.134	29	0
8	0.158	53	1
9	0.232	54	1

Preview the last 5 rows
df.tail()

	Pregnant	Glucose	Diastolic_BP	Skin_Fold	Serum_Insulin	BMI \
763	10	101.0	76.0	48.0	180.0	32.9
764	2	122.0	70.0	27.0	NaN	36.8
765	5	121.0	72.0	23.0	112.0	26.2
766	1	126.0	60.0	NaN	NaN	30.1
767	1	93.0	70.0	31.0	NaN	30.4

	Diabetes_Pedigree	Age	Class
763	0.171	63	0
764	0.340	27	0
765	0.245	30	0
766	0.349	47	1
767	0.315	23	0

Basic Summary Statistics

Use the describe() method to get basic statistics such as mean, median, minimum, maximum, and quartiles for each numeric column.

```
df.describe()
```

	Pregnant	Glucose	Diastolic_BP	Skin_Fold	Serum_Insulin
\count	768.000000	763.000000	733.000000	541.000000	394.000000
mean	3.845052	121.686763	72.405184	29.153420	155.548223
std	3.369578	30.535641	12.382158	10.476982	118.775855
min	0.000000	44.000000	24.000000	7.000000	14.000000
25%	1.000000	99.000000	64.000000	22.000000	76.250000
50%	3.000000	117.000000	72.000000	29.000000	125.000000
75%	6.000000	141.000000	80.000000	36.000000	190.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000

	BMI	Diabetes_Pedigree	Age	Class
count	757.000000	768.000000	768.000000	768.000000
mean	32.457464	0.471876	33.240885	0.348958
std	6.924988	0.331329	11.760232	0.476951
min	18.200000	0.078000	21.000000	0.000000
25%	27.500000	0.243750	24.000000	0.000000
50%	32.300000	0.372500	29.000000	0.000000
75%	36.600000	0.626250	41.000000	1.000000
max	67.100000	2.420000	81.000000	1.000000

Check Missing Values

It is crucial to check for missing data and understand how much of the data is missing in each column.

```
# Check for missing values
```

```
df.isnull().sum()
```

```
Pregnant      0
Glucose       5
Diastolic_BP  35
Skin_Fold     227
Serum_Insulin 374
BMI           11
Diabetes_Pedigree 0
Age           0
Class         0
dtype: int64
```

```
len(df)
```

768

Select rows by index position

```
# Select row by index
df.loc[3] # by label

Pregnant      1.000
Glucose       89.000
Diastolic_BP  66.000
Skin_Fold     23.000
Serum_Insulin 94.000
BMI           28.100
Diabetes_Pedigree 0.167
Age           21.000
Class         0.000
Name: 3, dtype: float64

df.iloc[1] # by position
```

Filter rows by condition

```
filtered = df[df["Age"] > 30]
filtered
```

Filter with multiple conditions

```
df[(df["Age"] > 30) & (df["Class"] == 1)]
df[["Age", "BMI"]]

   Age  BMI
0    50  33.6
1    31  26.6
2    32  23.3
3    21  28.1
4    33  43.1
...   ...  ...
763   63  32.9
764   27  36.8
765   30  26.2
766   47  30.1
767   23  30.4

[768 rows x 2 columns]
```