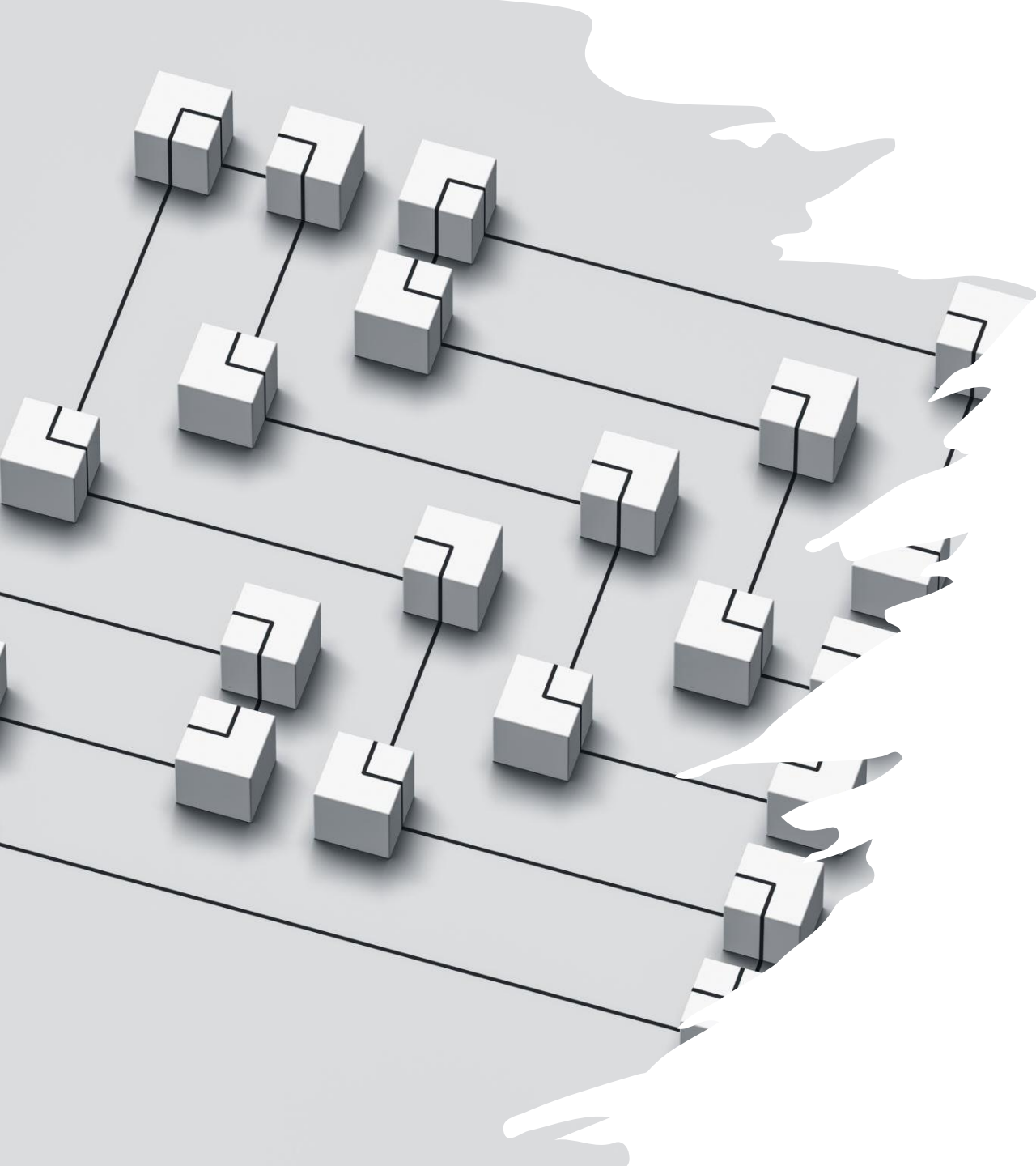




D.ATA SCIENCE INTER.NSHIP

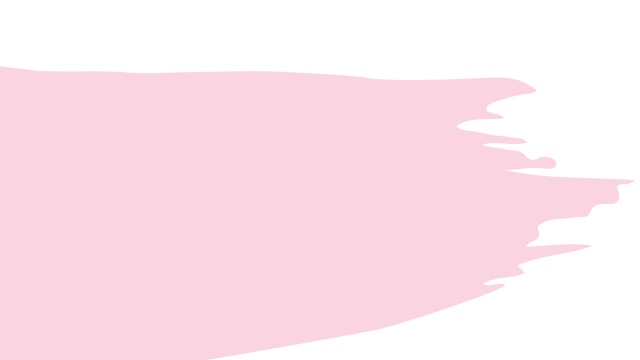
Session 2

HDLC TECHNOLOGIES



Create a DataFrame with Pandas

- A data frame is a structured representation of data.
- Let's define a data frame with 3 columns and 5 rows with fictional numbers:

A pink brushstroke is located in the top-left corner of the image, extending from the left edge and pointing towards the right.

```
import pandas as pd
d = {'col1': [1, 2, 3, 4, 7], 'col2': [4, 5, 6, 9, 5], 'col3': [7, 8, 12, 1, 11]}
df = pd.DataFrame(data=d)
print(df)
```

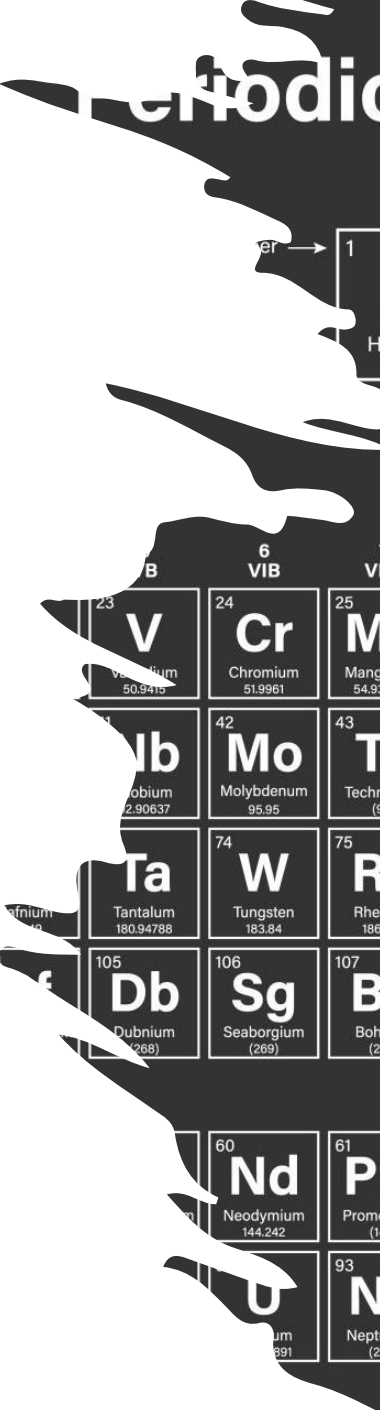


Output

		col1	col2	col3
1. row	0	1	4	7
2. row	1	2	5	8
3. row	2	3	6	12
4. row	3	4	9	1
5. row	4	7	5	11

Periodic Table of the Elements

Periodic Table of the Elements



The image shows a portion of the periodic table with a hand pointing to the element Hydrogen (H). The element box for Hydrogen is highlighted, showing its atomic number (1), symbol (H), name (Hydrogen), and atomic weight (1.008). Arrows point to the symbol and atomic weight labels. The periodic table includes elements from Hydrogen (1) to Oganesson (118), with the Lanthanide and Actinide series shown at the bottom.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
		B	C	N	O	F	Ne					Al	Si	P	S	Cl	Ar
3 Li Lithium 6.941	4 Be Beryllium 9.012	5 B Boron 10.81	6 C Carbon 12.011	7 N Nitrogen 14.007	8 O Oxygen 15.999	9 F Fluorine 18.998	10 Ne Neon 20.180	11 Na Sodium 22.990	12 Mg Magnesium 24.305	13 Al Aluminum 26.981	14 Si Silicon 28.085	15 P Phosphorus 30.973	16 S Sulfur 32.06	17 Cl Chlorine 35.45	18 Ar Argon 39.948	19 K Potassium 39.098	20 Ca Calcium 40.078
21 Sc Scandium 44.956	22 Ti Titanium 47.88	23 V Vanadium 50.942	24 Cr Chromium 51.996	25 Mn Manganese 54.938	26 Fe Iron 55.845	27 Co Cobalt 58.933	28 Ni Nickel 58.693	29 Cu Copper 63.546	30 Zn Zinc 65.38	31 Ga Gallium 69.723	32 Ge Germanium 72.630	33 As Arsenic 74.922	34 Se Selenium 78.96	35 Br Bromine 79.904	36 Kr Krypton 83.80	37 Rb Rubidium 85.468	38 Sr Strontium 87.62
39 Y Yttrium 88.906	40 Zr Zirconium 91.224	41 Nb Niobium 92.906	42 Mo Molybdenum 95.94	43 Tc Technetium (98)	44 Ru Ruthenium 101.07	45 Rh Rhodium 102.905	46 Pd Palladium 106.42	47 Ag Silver 107.868	48 Cd Cadmium 112.414	49 In Indium 114.818	50 Sn Tin 118.710	51 Sb Antimony 121.760	52 Te Tellurium 127.60	53 I Iodine 126.905	54 Xe Xenon 131.29	55 Cs Cesium 132.905	56 Ba Barium 137.327
57 La Lanthanum 138.905	58 Ce Cerium 140.12	59 Pr Praseodymium 140.908	60 Nd Neodymium 144.242	61 Pm Promethium (145)	62 Sm Samarium 150.36	63 Eu Europium 151.964	64 Gd Gadolinium 157.25	65 Tb Terbium 158.925	66 Dy Dysprosium 162.500	67 Ho Holmium 164.930	68 Er Erbium 167.259	69 Tm Thulium 168.934	70 Yb Ytterbium 173.045	71 Lu Lutetium 174.967	72 Hf Hafnium 178.49	73 Ta Tantalum 180.947	74 W Tungsten 183.84
75 Re Rhenium 186.207	76 Os Osmium 190.23	77 Ir Iridium 192.222	78 Pt Platinum 195.084	79 Au Gold 196.967	80 Hg Mercury 200.592	81 Tl Thallium 204.38	82 Pb Lead 207.2	83 Bi Bismuth 208.980	84 Po Polonium (209)	85 At Astatine (210)	86 Rn Radon (222)	87 Fr Francium (223)	88 Ra Radium (226)	89 Ac Actinium (227)	90 Th Thorium 232.037	91 Pa Protactinium 231.036	92 U Uranium 238.029
93 Np Neptunium (237)	94 Pu Plutonium (244)	95 Am Americium (243)	96 Cm Curium (247)	97 Bk Berkelium (247)	98 Cf Californium (251)	99 Es Einsteinium (252)	100 Fm Fermium (257)	101 Md Mendelevium (258)	102 No Nobelium (259)	103 Lr Lawrencium (260)	104 Rf Rutherfordium (261)	105 Db Dubnium (262)	106 Sg Seaborgium (266)	107 Bh Bohrium (264)	108 Hs Hassium (277)	109 Mt Meitnerium (268)	110 Ds Darmstadtium (271)
111 Rg Roentgenium (272)	112 Cn Copernicium (285)	113 Nh Nihonium (286)	114 Fl Flerovium (289)	115 Mc Moscovium (289)	116 Lv Livermorium (293)	117 Ts Tennessine (294)	118 Og Oganesson (294)	119 Uue Ununennium (295)	120 Uub Unbibium (298)	121 Uut Untrium (301)	122 Uuq Unquadium (304)	123 Uup Unpentium (307)	124 Uuh Unhexium (310)	125 Uus Unseptium (313)	126 Uuq Unoctium (316)	127 Uuh Unnonium (319)	128 Uuo Unnilium (321)

[illegible]

Count the number of rows:

- `count_row = df.shape[0]`
`print(count_row)`

Data Science Functions

• The Sports Watch Data Set

Duration	Average_Pulse	Max_Pulse	Calorie_Burnage	Hours_Work	Hours_Sleep
30	80	120	240	10	7
30	85	120	250	10	7
45	90	130	260	8	7
45	95	130	270	8	7
45	100	140	280	0	7
60	105	140	290	7	8
60	110	145	300	7	8
60	115	145	310	8	8
75	120	150	320	0	8
75	125	150	330	8	8

- The data set above consists of 6 variables, each with 10 observations:
- **Duration** - How long lasted the training session in minutes?
- **Average_Pulse** - What was the average pulse of the training session? This is measured by beats per minute
- **Max_Pulse** - What was the max pulse of the training session?
- **Calorie_Burnage** - How much calories were burnt on the training session?
- **Hours_Work** - How many hours did we work at our job before the training session?
- **Hours_Sleep** - How much did we sleep the night before the training session?

The max() function

- ```
Average_pulse_max =
max(80, 85, 90, 95, 100, 105, 110, 115, 120, 125)

print (Average_pulse_max)
```



# The min() function

- `Average_pulse_min =`  
`min(80, 85, 90, 95, 100, 105, 110, 115, 120, 125)`  
  
`print (Average_pulse_min)`

# The mean() function

- `import numpy as np`

```
Calorie_burnage
= [240, 250, 260, 270, 280, 290, 300, 310, 320,
 330]
```

```
Average_calorie_burnage
= np.mean(Calorie_burnage)
```

```
print(Average_calorie_burnage)
```



# Data Science - Data Preparation

- Before analyzing data, a Data Scientist must extract the data, and make it clean and valuable.
  - Extract and Read Data With Pandas
  - Before data can be analyzed, it must be imported/extracted.
- 
- ```
import pandas as pd
```



```
health_data =  
pd.read_csv("data.csv")
```



```
print(health_data)
```
 - ```
print(health_data.head())
```

# Data Cleaning

- Look at the imported data. As you can see, the data are "dirty" with wrongly or unregistered values:
- There are some blank fields
- Average pulse of 9 000 is not possible
- 9 000 will be treated as non-numeric, because of the space separator
- One observation of max pulse is denoted as "AF", which does not make sense
- So, we must clean the data in order to perform the analysis.

|    | Duration | Average_Pulse | Max_Pulse | Calorie_Burnage | Hours_Work | Hours_Sleep |
|----|----------|---------------|-----------|-----------------|------------|-------------|
| 0  | 30.0     | 80            | 120       | 240.0           | 10.0       | 7.0         |
| 1  | 45.0     | 85            | 120       | 250.0           | 10.0       | 7.0         |
| 2  | 45.0     | 90            | 130       | 260.0           | 8.0        | 7.0         |
| 3  | 60.0     | 95            | 130       | 270.0           | 8.0        | 7.0         |
| 4  | 60.0     | 100           | 140       | 280.0           | 0.0        | 7.0         |
| 5  | NaN      | NaN           | NaN       | NaN             | NaN        | NaN         |
| 6  | 60.0     | 105           | 140       | 290.0           | 7.0        | 8.0         |
| 7  | 60.0     | 110           | 145       | 300.0           | 7.0        | 8.0         |
| 8  | 45.0     | NaN           | AF        | NaN             | 8.0        | 8.0         |
| 9  | 45.0     | 115           | 145       | 310.0           | 8.0        | 8.0         |
| 10 | 60.0     | 120           | 150       | 320.0           | 0.0        | 8.0         |
| 11 | 60.0     | 9 000         | 130       | NaN             | NaN        | 8.0         |
| 12 | 45.0     | 125           | 150       | 330.0           | 8.0        | 8.0         |

# Remove Blank Rows

- `health_data.dropna(axis=0,inplace=True)`

```
print(health_data)
```

|    | Duration | Average_Pulse | Max_Pulse | Calorie_Burnage | Hours_Work | Hours_Sleep |
|----|----------|---------------|-----------|-----------------|------------|-------------|
| 0  | 30.0     | 80            | 120       | 240.0           | 10.0       | 7.0         |
| 1  | 45.0     | 85            | 120       | 250.0           | 10.0       | 7.0         |
| 2  | 45.0     | 90            | 130       | 260.0           | 8.0        | 7.0         |
| 3  | 60.0     | 95            | 130       | 270.0           | 8.0        | 7.0         |
| 4  | 60.0     | 100           | 140       | 280.0           | 0.0        | 7.0         |
| 6  | 60.0     | 105           | 140       | 290.0           | 7.0        | 8.0         |
| 7  | 60.0     | 110           | 145       | 300.0           | 7.0        | 8.0         |
| 9  | 45.0     | 115           | 145       | 310.0           | 8.0        | 8.0         |
| 10 | 60.0     | 120           | 150       | 320.0           | 0.0        | 8.0         |
| 12 | 45.0     | 125           | 150       | 330.0           | 8.0        | 8.0         |

# Data Categories

- To analyze data, we also need to know the types of data we are dealing with.
- Data can be split into three main categories:
  - 1. Numerical** - Contains numerical values. Can be divided into two categories:
    1. Discrete: Numbers are counted as "whole". Example: You cannot have trained 2.5 sessions, it is either 2 or 3
    2. Continuous: Numbers can be of infinite precision. For example, you can sleep for 7 hours, 30 minutes and 20 seconds, or 7.533 hours
  - 2. Categorical** - Contains values that cannot be measured up against each other. Example: A color or a type of training
  - 3. Ordinal** - Contains categorical data that can be measured up against each other. Example: School grades where A is better than B and so on
- By knowing the type of your data, you will be able to know what technique to use when analyzing them.

# Data Types

- `print(health_data.info())`

We see that this data set has two different types of data:

- Float64
- Object

We cannot use objects to calculate and perform analysis here. We must convert the type object to float64 (float64 is a number with a decimal in Python).

```
Data columns (total 6 columns):
Column Non-Null Count Dtype
--- -
0 Duration 12 non-null float64
1 Average_Pulse 11 non-null object
2 Max_Pulse 12 non-null object
3 Calorie_Burnage 10 non-null float64
4 Hours_Work 11 non-null float64
5 Hours_Sleep 12 non-null float64
dtypes: float64(4), object(2)
```

# *Convert object into float*

```
• health_data["Average_Pulse"] =
 health_data['Average_Pulse'].astype(float)
)
health_data["Max_Pulse"]
= health_data["Max_Pulse"].astype(float)
print (health_data.info())
```

```
Data columns (total 6 columns):
Column Non-Null Count Dtype
--- ----- -
0 Duration 10 non-null float64
1 Average_Pulse 10 non-null float64
2 Max_Pulse 10 non-null float64
3 Calorie_Burnage 10 non-null float64
4 Hours_Work 10 non-null float64
5 Hours_Sleep 10 non-null float64
dtypes: float64(6)
```



# Analyze the Data

- `print(health_data.describe())`

|       | Duration | Average_Pulse | Max_Pulse | Calorie_Burnage | Hours_Work | Hours_Sleep |
|-------|----------|---------------|-----------|-----------------|------------|-------------|
| Count | 10.0     | 10.0          | 10.0      | 10.0            | 10.0       | 10.0        |
| Mean  | 51.0     | 102.5         | 137.0     | 285.0           | 6.6        | 7.5         |
| Std   | 10.49    | 15.4          | 11.35     | 30.28           | 3.63       | 0.53        |
| Min   | 30.0     | 80.0          | 120.0     | 240.0           | 0.0        | 7.0         |
| 25%   | 45.0     | 91.25         | 130.0     | 262.5           | 7.0        | 7.0         |
| 50%   | 52.5     | 102.5         | 140.0     | 285.0           | 8.0        | 7.5         |
| 75%   | 60.0     | 113.75        | 145.0     | 307.5           | 8.0        | 8.0         |
| Max   | 60.0     | 125.0         | 150.0     | 330.0           | 10.0       | 8.0         |

- **Count** - Counts the number of observations

- **Mean** - The average value

- **Std** - Standard deviation

- **Min** - The lowest value

- **25%, 50%** and **75%** are percentiles.

- **Max** - The highest value