# Data analysis portfolio

## Spring 2

**Name: Zeina Essam Mahmoud Ali**

**ID:1120157982**

**Group: Data analytics C2 DEPI-EUI**

# Task 4

## Summary of Cleaning Steps:

**1- Standardized Product Names:**

- Removed extra spaces and any additional characters not part of the product name using Trim ()

**2- Normalized Email Addresses:**

- Converted all email addresses to lowercase for consistency using Lower()

**3- Handled Missing Values:**

- Filled missing values in quarterly sales columns with 0.
- Filled missing values in the "Percentage" column with the median value.

**4- Converted Data Types:**

- Ensured all numerical data was in the correct format for calculations.

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Product | User_Name | Customer email | Qtr 1 | Qtr 2 | Qtr 3 | Qtr 4 | Q1/Q2 | Percentage |
| 2 | Alice | Anton | ANTON@gmail.com | $ - | $ 702.00 | $ - | $ - | 0 | 78% |
| 3 | Alice | Bergs | BERGS@gmail.com | $ 312.00 | $ 1.00 | $ - | $ - | 312 | 14% |
| 4 | Alice | Bolid | BOLID@gmail.com | $ - | $ 1.00 | $ - | $ 1,170.00 | 0 | 43% |
| 5 | Alice | Bottm | BOTTM@gmail.com | $ 1,170.00 | $ 1.00 | $ - | $ - | 1170 | 86% |
| 6 | Alice | Ernsh | ERNSH@gmail.com | $ 1,123.20 | $ 1.00 | $ - | $ - | 1123.2 | 60% |
| 7 | Alice | Godos | GODOS@gmail.com | $ - | $ 1.00 | $ - | $ - | 0 | 71% |
| 8 | Alice | Hungc | HUNGC@gmail.com | $ 62.40 | $ 1.00 | $ - | $ - | 62.4 | 75% |
| 9 | Alice | Picco | PICCO@gmail.com | $ - | $ 1,560.00 | $ 936.00 | $ - | 0 | 51% |
| 10 | Alice | Rattc | RATTC@gmail.com | $ - | $ 592.80 | $ - | $ - | 0 | 74% |
| 11 | Alice | Reggc | REGGC@gmail.com | $ - | $ 1.00 | $ - | $ 741.00 | 0 | 66% |
| 12 | Alice | Reggc | REGGC@gmail.com | $ - | $ 1.00 | $ - | $ 741.00 | 0 | 48% |
| 13 | Alice | Savea | SAVEA@gmail.com | $ - | $ 1.00 | $ - | $ 789.75 | 0 | 8% |
| 14 | Alice | Seves | SEVES@gmail.com | $ - | $ 877.50 | $ - | $ - | 0 | 45% |
| 15 | Alice | Whitc | WHITC@gmail.com | $ - | $ 1.00 | $ - | $ 780.00 | 0 | 53% |
| 16 | Aniseed | Alfki | ALFKI@gmail.com | $ - | $ 1.00 | $ - | $ 60.00 | 0 | 52% |
| 17 | Aniseed | Bottm | BOTTM@gmail.com | $ - | $ 1.00 | $ - | $ 200.00 | 0 | 6% |
| 18 | Aniseed | Ernsh | ERNSH@gmail.com | $ - | $ 1.00 | $ - | $ 180.00 | 0 | 89% |
| 19 | Aniseed | Linod | LINOD@gmail.com | $ 544.00 | $ 1.00 | $ - | $ - | 544 | 20% |
| 20 | Aniseed | Quick | QUICK@gmail.com | $ - | $ 600.00 | $ - | $ - | 0 | 50% |
| 21 | Aniseed | Vaffe | VAFFE@gmail.com | $ - | $ 1.00 | $ 140.00 | $ - | 0 | 15% |
| 22 | Boston | Anton | ANTON@gmail.com | $ - | $ 165.60 | $ - | $ - | 0 | 19% |
| 23 | Boston | Bergs | BERGS@gmail.com | $ - | $ 920.00 | $ - | $ - | 0 | 83% |
| 24 | Boston | Bonap | BONAP@gmail.com | $ - | $ 248.40 | $ 524.40 | $ - | 0 | 69% |
| 25 | Boston | Bottm | BOTTM@gmail.com | $ 551.25 | $ 1.00 | $ - | $ - | 551.25 | 11% |
| 26 | Boston | Bsbev | BSBEV@gmail.com | $ 147.00 | $ 1.00 | $ - | $ - | 147 | 52% |
| 27 | Boston | Frans | FRANS@gmail.com | $ - | $ 1.00 | $ - | $ 18.40 | 0 | 78% |
| 28 | Boston | Hilaa | HILAA@gmail.com | $ - | $ 92.00 | $ 1,104.00 | $ - | 0 | 79% |
| 29 | Boston | Lazyk | LAZYK@gmail.com | $ 147.00 | $ 1.00 | $ - | $ - | 147 | 16% |
| 30 | Boston | Lehms | LEHMS@gmail.com | $ - | $ 515.20 | $ - | $ - | 0 | 81% |
| 31 | Boston | Magaa | MAGAA@gmail.com | $ - | $ 1.00 | $ - | $ 55.20 | 0 | 55% |
| 32 | Boston | Ottik | OTTIK@gmail.com | $ - | $ 1.00 | $ 368.00 | $ - | 0 | 52% |

# Task 5

The Road Accident Casualties Data Dashboard is a dynamic tool designed to offer comprehensive insights into the patterns and factors contributing to road accident casualties.

## Key Features

1. **Summary Statistics:**
    - **Total Casualties:** A chart displaying the overall number of casualties.
    - **Fatal Casualties:** A chart illustrating the number of fatalities resulting from road accidents.
    - **Serious Casualties:** A chart showing the number of severe injuries sustained in accidents.
    - **Slight Casualties:** A chart depicting the number of minor injuries.
    - **Accidents by Car:** A chart presenting the total number of accidents involving cars.
2. **Timeline Analysis:**
    - **Accidents by Year:** Trends over multiple years.
    - **Accidents by Month:** Monthly variations to identify peak periods for accidents.
3. **Geographical Analysis:**
    - **Distribution by Region/City:** Maps showing casualties in rural vs. urban areas.
4. **Demographic Analysis:**
    - **Vehicle Type:** Casualties involving various vehicle types (cars, motorcycles, bicycles, trucks).
5. **Contributing Factors:**
    - **Road Type:** Analysis of accidents on different road types (e.g., slip roads, one-way streets).
    - **Road Surface:** Effects of road conditions (wet, dry, icy, etc.) on accidents.
    - **Light Conditions:** Breakdown of accidents occurring during daylight, twilight, and nighttime.

## Dashboard Sections and Insights

### *1. Summary Statistics*

The summary section provides a quick overview of key metrics related to road accidents, including:

- **Total Casualties:** A comprehensive chart showcasing the total number of casualties.
- **Fatal, Serious, and Slight Casualties:** Individual charts for each casualty severity level, offering detailed insights.
- **Accidents by Car:** A chart illustrating the prevalence of car-involved accidents.2. Timeline Analysis

- **Yearly Trends:** Line charts depicting the annual number of accidents, revealing significant trends and changes.

### *3. Geographical Analysis*

This section uses geographical visualizations to display accident data spatially:

- **Casualty Distribution:** par chart comparing the distribution of casualties in rural and urban areas.

### *4. Demographic Analysis*

Demographic charts provide valuable insights into affected population segments:

- **Vehicle Type Analysis:** Detailed analysis of casualties involving various vehicle types, including cars, motorcycles, bicycles, and trucks.

### *5. Contributing Factors*

This section explores conditions and behaviors contributing to casualties:

- **Road Type:** Charts showing the distribution of accidents on different types of roads, such as slip roads and one-way streets.
- **Road Surface:** Analysis of accidents under various road conditions (wet, dry, icy, etc.).
- **Light Conditions:** Breakdown of accidents occurring under different light conditions, including daylight, twilight, and nighttime.
- **Primary Causes:** Bar charts or tree maps identifying common causes of accidents, such as speeding and distracted driving.

## Slicer and Timeline Management

### *Road Type Slicer*

An interactive slicer allows users to filter the dashboard data by road type, providing a focused view on how different road types impact accident and casualty rates.

### *Timeline Management*

The dashboard includes timeline management features that allow users to adjust the time frame of the data being viewed. This includes:

- **Yearly and Monthly Filters:** Users can select specific years or months to analyze trends within those periods.
- **Time Slider:** A dynamic slider for users to easily navigate through different time ranges, facilitating detailed temporal analysis.

# The Dashboard:

# Task 6

This application captures essential personal details efficiently and clearly displays the collected information and perform basic arithmetic operations.

**Key Features:**

## User Information Collection:

- **Name Input:** Users are prompted to enter their name.
- **Age Input:** Users are prompted to enter their age.
- **Email Input:** Users are prompted to enter their Gmail address.
- These inputs are captured using simple input prompts and displayed back to the user.

Arithmetic Operations:

- Users can enter two numbers.
- The application automatically calculates and displays the following:

    - Sum: Total of the two numbers.
    - Difference: Result of subtracting the second number from the first.
    - Product: Result of multiplying the two numbers.
    - Integer Division: Quotient of the first number divided by the second, ignoring any remainder.
    - Remainder: The remainder when the first number is divided by the second.

## Technologies Used:

**Python Input Handling:**

- Basic arithmetic operations using Python's built-in operators (+, -, *, //, %).
- Used the input () function to capture user inputs.
- Displayed the collected information using formatted output (f-strings).

```
[ ] name=input("What is your name:")
    Email=input("What is your Email:")
    age=input("What is your Age:")
    print("your Info:\t"+name+"\t"+Email+"\t"+age)

    What is your name:Zeina
    What is your Email:zeinaessa.@gmail.com
    What is your Age:23
    your Info:      Zeina   zeinaessa.@gmail.com    23

    x =int(input("insert number first number:"))
    y=int(input("insert number first number:"))
    sum1=x+y
    sub=x-y
    multi=x*y
    div=x/y
    div2=x//y
    remain=x%y

    print(f"sum = {sum1}\nsub= {sub}\nmult={multi}\ndiv={div}\ninteger_devision={div2}\nremaning={remain}")

    insert number first number:2
    insert number first number:1
    sum = 3
    sub= 1
    mult=2
    div=2.0
    integer_devision=2
    remaning=0
```

# Task7

**Overview:** I developed an interactive application in Python designed to calculate the average of five numbers entered by the user.

**Key Features:**

- **Number Input:** Users are prompted to enter five numbers.
- **Average Calculation:** The application calculates and displays the average of the entered numbers.

**Technologies Used:**

- **Python Input Handling and Calculation:**
  - Used the input() function to capture user inputs.
  - Calculated the average using basic arithmetic operations.
  - Displayed the result using formatted output (f-strings).

```
numbers=[]
count=5
while len(numbers)< count :
  x=int(input("Enter numbers: \n"))

  if x< 0:
    print("Enter a postive number")
  else:
    numbers.append(x)

print("the avarge is",sum(numbers)/len(numbers))
```

```
Enter numbers:
3
Enter numbers:
4
Enter numbers:
2
Enter numbers:
4
Enter numbers:
5
the avarge is 3.6
```

**2- Interactive Number Categorization Application in Python**

**Overview:** I developed an interactive application in Python that categorizes numbers based on user input until a zero is entered. This tool categorizes numbers as even-positive, odd-positive, even-negative, or odd-negative, and provides a summary of these categories.

**Key Features:**

- **Number Input:** Users can enter numbers continuously until they enter zero to stop.

- **Number Categorization:** The application categorizes the entered numbers into:
  - Even Positive
  - Odd Positive
  - Even Negative
  - Odd Negative
- **Summary Display:** Displays a summary of the categorized numbers once the user exits by entering zero.

**Technologies Used:**

- **Python Input Handling and Conditional Logic:**
  - Used the input () function to capture user inputs.
  - Employed conditional statements to categorize numbers.
  - Lists

```
odd_postivie=0
even_negative=0
odd_negative=0

while True:
  x=int(input("Enter a number:"))

  if x == 0:
    break
  elif x>0 and x%2==0:
    even_postivie +=1
  elif x>0 and x%2 !=0:
    odd_postivie +=1
  elif x<0 and x%2==0:
    even_negative +=1
  elif x<0 and x%2 !=0:
    odd_negative +=1

print("The even postive numbers =",even_postivie)
print("The odd postive numbers =",odd_postivie)
print("The even negative numbers =",even_negative)
print("The odd negative numbers =",odd_negative)
```

```
Enter a number:1
Enter a number:-1
Enter a number:2
Enter a number:-2
Enter a number:0
The even postive numbers = 1
The odd postive numbers = 1
The even negative numbers = 1
The odd negative numbers = 1
```

# Task 8

## 1- User ID Validation:

**Overview:** I developed an interactive application in Python designed to validate user IDs based on specific criteria. The tool ensures that the ID starts with "AB", has a length of 7 characters, and the last four characters are numbers.

**Key Features:**

- **User ID Validation:**
  - **ID Input:** Users are prompted to enter their ID.
  - **Validation Criteria:**
    - The ID must start with "AB".
    - The ID must be exactly 7 characters long.
    - The last four characters must be numbers.
  - **Feedback:** The application provides immediate feedback on whether the entered ID is valid or not.

**Technologies Used:**

- **Python String Handling and Validation:**
  - Used the input() function to capture user input.
  - Employed string methods and conditional statements to validate the ID.
  - Provided feedback using formatted output (f-strings).

```
for i in range(1,11):
    for j in range (i,11):
        print(f"{i}x {j} =",i*j)
    print("..............")
```

```
1x 8 = 8
1x 9 = 9
1x 10 = 10
..............
2x 2 = 4
2x 3 = 6
2x 4 = 8
2x 5 = 10
2x 6 = 12
2x 7 = 14
2x 8 = 16
2x 9 = 18
2x 10 = 20
..............
3x 3 = 9
3x 4 = 12
3x 5 = 15
3x 6 = 18
3x 7 = 21
3x 8 = 24
3x 9 = 27
3x 10 = 30
..............
4x 4 = 16
4x 5 = 20
4x 6 = 24
4x 7 = 28
4x 8 = 32
```

## 2- Interactive Multiplication Table Application in Python:

**Overview:** I developed an interactive application in Python that prints the multiplication table from 1 to 10, ensuring that each multiplication result is printed only once.

**Key Features:**

- **Multiplication Table:**
  - **Unique Results:** Ensures that each multiplication result is printed only once.
  - **Clear Display:** Presents the multiplication table in a clear and readable format.

**Technologies Used:**

- **Python Loops and Conditional Logic:**
  - Used nested loops to generate the multiplication table.
  - Employed a set to track and print unique results only.

```
[10]
     id=input("Enter your ID:")

     if not id.startswith("AB"):
       print("the ID has to start with AB ")
     elif len(id) != 7:
       print("ID has to be 7 digits")
     elif not id[-4:].isdigit():
       print("ID has to end with 4 digits")
     else:
       print("ID is valid")
```

```
Enter your ID:ABx1234
ID is valid
```

# Task 9

## 1- Interactive FizzBuzz Application in Python

**Overview:**

I developed an interactive application in Python that implements the classic FizzBuzz problem. replacing certain multiples with the words "Fizz", "Buzz", or "FizzBuzz" based on specific criteria.

**Key Features:**

- **FizzBuzz Logic:**
  - **Multiples of 3:** Replaces numbers that are multiples of 3 with "Fizz".
  - **Multiples of 5:** Replaces numbers that are multiples of 5 with "Buzz".
  - **Multiples of 3 and 5:** Replaces numbers that are multiples of both 3 and 5 with "FizzBuzz".
- **Clear Display:** Presents the FizzBuzz results in a clear and readable format.

**Technologies Used:**

- **Python Loops and Conditional Logic:**
  - Used the `input()` function to capture user input.
  - Employed conditional statements to determine the appropriate replacement for each number.
  - Displayed the results using formatted output (print statements).

Certainly! Here's a description and implementation of an interactive Python function to check if a number is prime. The user is prompted to enter a number, and the function determines if the number is prime or not.

```
[17] def print_numbers(n):
       if n%3==0 and n%5==0:
         print("FizzBuzz")
       elif n%3==0:
         print("Fizz")
       elif n%5==0:
         print("Buzz")
       else:
         print(n)

     print_numbers(15)
```

```
→ FizzBuzz
```

## 2- Interactive Prime Number Checker Application in Python

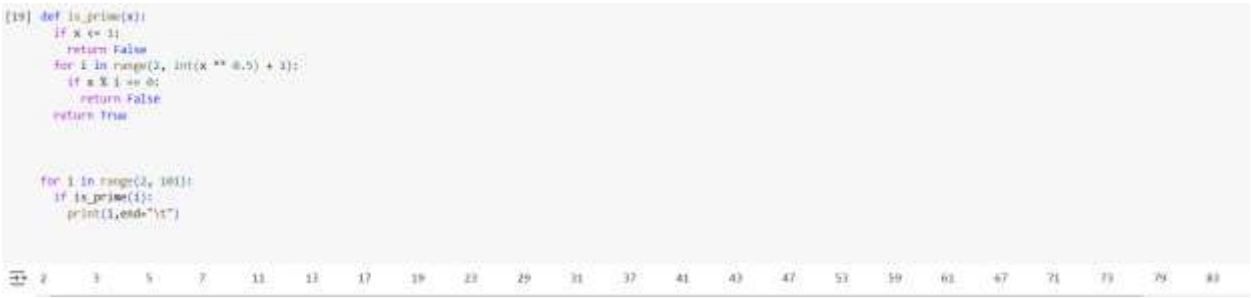
**Overview:** I developed an interactive application in Python that checks if a given number is a prime number. The tool prompts the user to enter a number and then determines whether the entered number is prime.

**Key Features:**

- **User Input:**
  - Prompts the user to enter a number.
  - Ensures that the user enters a valid positive integer.
- **Prime Number Logic:**
  - Checks if the entered number is a prime number.

**Technologies Used:**

- **Python Input Handling and Prime Checking Logic:**
  - Used the input() function to capture user input.
  - Employed a function to check if a number is prime using basic mathematical principles.
  - Displayed the result using formatted output (print statements)



3- **Sublist Checker Function in Python**

**Overview:** I developed a Python function that checks if one list is a sublist of another. The tool takes two lists as input and determines if the second list appears as a contiguous sequence within the first list.

**Key Features:**

- **Sublist Checking Logic:**
  - **Input Handling:** Takes two lists as input parameters.
  - **Sublist Verification:** Checks if the second list is a contiguous sublist of the first list.
  - **Output:** Returns `True` if the second list is a sublist of the first, `False` otherwise.

**Technologies Used:**

- **Python List Handling and Slicing:**
  - Implemented a function to encapsulate the logic for reusability and clarity.

```
def prefix(list1, list2):
  result=[]
  for i in range(len(list1)):
    if list1[i]==list2[i]:
      result.append(i)
      return True
    else:
      return False

prefix([1, 2, 2, 3], [1, 2, 3, 4])
```

```
True
```

## 4- Union of Two Lists Without Duplicates Function in Python

**Overview:** I developed a Python function that computes the union of two lists, ensuring that each element appears only once in the resulting list. The tool takes two lists as input and returns a new list containing all unique elements from both lists.

**Key Features:**

- **Union Logic:**
  - **Input Handling:** Takes two lists as input parameters.
  - **Combination and Deduplication:** Combines elements from both lists and removes duplicates.
  - **Output:** Returns a list containing the union of the two input lists without any duplicates.

**Technologies Used:**

- **Python List Handling and Set Operations:**
  - Used Python's set data structure to handle deduplication and compute the union.
  - Implemented a function to encapsulate the logic for reusability and clarity.

## 5- Flatten Nested List Function in Python

**Overview:** I developed a Python function that flattens a nested list. The tool takes a nested list as input and returns a new list with all elements extracted and included in a single, flat list.

**Key Features:**

- **Flattening Logic:**
  - **Input Handling:** Takes a nested list as input.
  - **Output:** Returns a flat list containing all elements from the nested list.

**Technologies Used:**

- **Python List Handling and Recursion:**
  - Implemented a function to encapsulate the logic for reusability and clarity.

```
def union(list1,list2):
  l=[]
  for i in list1 and list2:
    if i not in l:
      l.append(i)

  return l


union([1,2,3,2],[2,4,3])
```

```
[2, 4, 3]
```

```
[21] def flatten(list1):
       result = []
       for i in list1:
         for j in i:
           result.append(j)
       return result


     flatten([[1,2,3,2],[2,4,3]])
```

```
[1, 2, 3, 2, 2, 4, 3]
```