Advanced Python

for Neuroscience

Saeed Mohagheghi

2025 | ۱۴۰۴

ویژگی‌های دوره

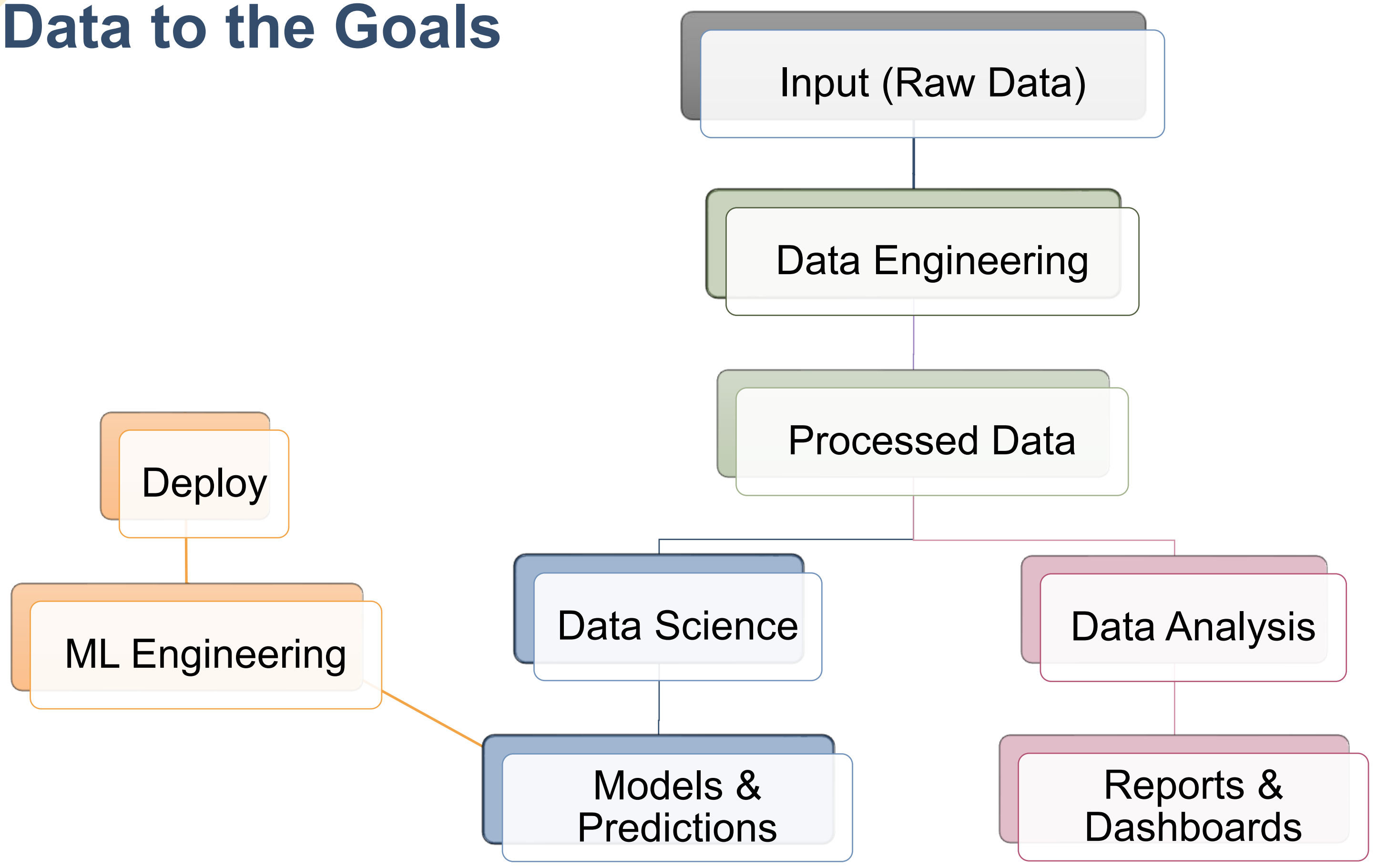
- پروژه محور
- کار با داده‌های واقعی
- مفاهیم پیشرفته پایتون
- کدنویسی زنده (Live Coding)
- کدنویسی با هوش مصنوعی (Vibe Coding)
- ...

پیش نیازها

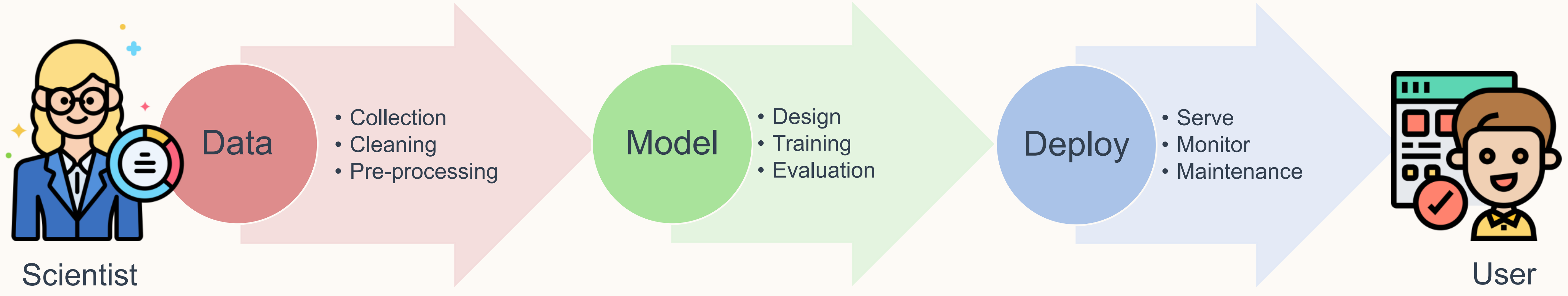
- پایتون مقدماتی
  - متغیرها / دستورات شرطی / حلقه‌ها / توابع / کلاس‌ها
- کتابخانه‌های numpy / pandas / matplotlib
- آشنایی با محیط Notebook و Google Colab
- زبان انگلیسی (در حد جستجو و استفاده از هوش مصنوعی)
- توانایی حل مساله
- ...

The Workflow

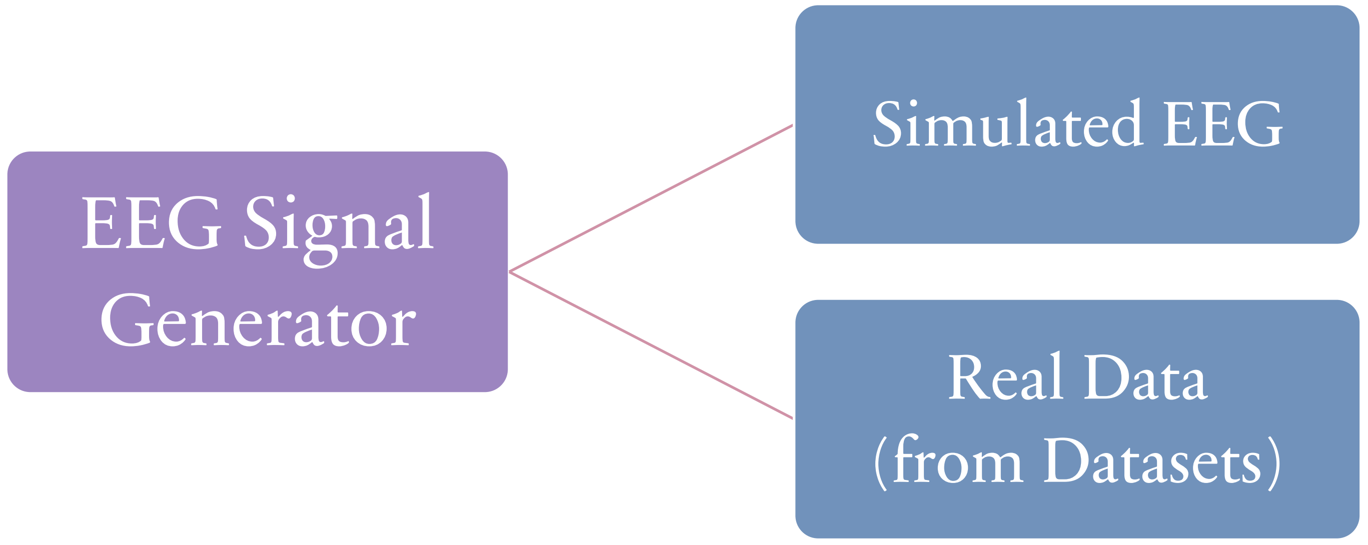
From the Data to the Goals



Data Science Pipeline



پروژه اول (Data)



سبک‌های کدنویسی

• **Procedural**	• Top-down sequence of instructions and reusable procedures
• **Functional**	• Functions as primary citizens, immutable data, and data transformation In Python: first-class functions, map, filter, and lambda expressions
• **Object-oriented (OOP)**	• Encapsulating data and the functions (methods) into self-contained objects

programming_paradigms.ipynb

*args / **kwargs

- قابلیت تعریف تعداد نامعلوم پارامتر ورودی برای توابع

- ***args**:
  - Allows a function to accept a variable number of **positional arguments**.
  - The arguments are passed as a tuple.

- ****kwargs**:
  - Allows a function to accept a variable number of **keyword arguments**.
  - The arguments are passed as a dictionary.

مثال‌های *args و **kwargs

```python
def sum_numbers(*args):
    return sum(args)

result = sum_numbers(1, 2, 3, 4)
# result is 10
```

```python
def print_info(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

print_info(name="Alice", age=30)
```

Output:
```
name: Alice
age: 30
```

```python
def mixed_function(*args, **kwargs):
    print("Positional arguments:", args)
    print("Keyword arguments:", kwargs)

mixed_function(1, 2, name="Bob", age=25)
```

Output:
```
Positional arguments: (1, 2)
Keyword arguments: {'name': 'Bob', 'age': 25}
```

https://book.pythontips.com/en/latest/args_and_kwargs.html

Generators in Python

- تولید خروجی در حین اجرا (بدون نیاز به ذخیره در حافظه)
- Values are generated on-the-fly and only when requested
- Defined using a function with the yield statement
- Retains its state between calls,

  allowing it to resume where it left off

```
def countdown(n):
    while n > 0:
        yield n
        n -= 1

# Usage
for number in countdown(5):
    print(number)
# Output:
# 5
# 4
# 3
# 2
# 1
```

•Use `next(generator)`to manually retrieve the next value from a generator.

Generators in Python

- Advantages of Generators
  - **Memory Efficiency**: Ideal for working with large datasets or streams of data.
  - **Improved Performance**: Avoids the overhead of creating and storing an entire list.
  - **Pipelining**: Can be used to create pipelines for processing data in stages.

- Use Cases
  - Reading large files line by line.
  - Generating infinite sequences (e.g., Fibonacci numbers).
  - Processing data streams (e.g., web scraping).

https://book.pythontips.com/en/latest/generators.html

Function Decorators

- تغییر رفتار یا افزودن عملیات به توابع، بدون تغییر تابع

- A **decorator** is a special type of function that **modifies the behavior of another function**.
- Addition of functionality to existing code in a clean and readable way.

```python
def decorator(func):
    def wrapper():
        print("Before calling the function.")
        func()
        print("After calling the function.")
    return wrapper
```

```python
@decorator
def target_function():
    print("Hello World")
```

```python
target_function():
```

```
Output:
Before calling the function.
Hello, World!
After calling the function.
```

https://book.pythontips.com/en/latest/decorators.html

Asynchronous Programming in Python

- A method of **parallel programming** that allows tasks to run concurrently without blocking the main thread
- **asyncio**: A library to write concurrent code using the async and await syntax.

- **Advantages of Asynchronous Programming**
  - **Improved Responsiveness**: Ideal for applications with I/O-bound operations (e.g., network requests, file I/O).
  - **Concurrency**: Easily manage multiple tasks without blocking.
  - **Better Resource Utilization**: Allows for multiple tasks to run in overlapping time periods, maximizing CPU usage.

مهم‌ترین توابع asyncio

- **asyncio.Queue**
  - A FIFO queue designed for use with coroutines.
  - Useful for task synchronization and communication between producers and consumers.
- **asyncio.create_task()**
  - Schedules the execution of a coroutine and returns a Task object.
  - Allows multiple coroutines to run concurrently.
- **asyncio.gather()**
  - Runs multiple coroutines concurrently and waits for all of them to complete.
  - Returns results in the order of the input coroutines.
- **asyncio.run()**
  - A high-level function to run the main entry point of an asynchronous program.
  - It handles the event loop and ensures proper cleanup.

async_example.ipynb

Thank You

Saeed Mohagheghi

Daneshjoy.ir@gmail.com

https://github.com/DaneshJoy/

https://www.linkedin.com/in/saeed-mohagheghi