Giai đoạn 1: Làm quen với lớp và đối tượng

In [4]:
```python
# Bài 1.1: Tạo lớp Student có các thuộc tính: name, id, major. Tạo 3 đối tượng v
# in ra thông tin
class Student:
    def __init__(self, name, id, major):
        self.name = name
        self.id = id
        self.major = major
    def print_info(self):
        print(f"Name: {self.name}, Id: {self.id}, Major: {self.major}")

s1 = Student("An", "001", "CS")
s2 = Student("Bình", "002", "DS")
s3 = Student("Chung", "003", "AI")

students = [s1, s2, s3]
for stu in students:
    stu.print_info()
```

```
Name: An, Id: 001, Major: CS
Name: Bình, Id: 002, Major: DS
Name: Chung, Id: 003, Major: AI
```

In [5]:
```python
# Bài 1.2: Viết phương thức __init__() cho lớp Student để khởi tạo dữ
# liệu khi tạo đối tượng
class Student:
    def __init__(self, name, id, major):
        self.name = name
        self.id = id
        self.major = major
```

In [7]:
```python
# Bài 1.3: Thêm phương thức display() để hiển thị thông tin sinh viên
class Student:
    def __init__(self, name, id, major):
        self.name = name
        self.id = id
        self.major = major
    def display(self):
        print(f"Name: {self.name}, Id: {self.id}, Major: {self.major}")
```

In [14]:
```python
# Bài 1.4: Viết lớp Circle có thuộc tính radius, phương thức area() và perimeter
import math
class Circle:
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return math.pi*self.radius**2
    def perimeter(self):
        return 2*math.pi*self.radius

c1 = Circle(2)
print(f"Radius is 2: area= {c1.area()}, perimeter= {c1.perimeter()}")
```

```
Radius is 2: area= 12.566370614359172, perimeter= 12.566370614359172
```

In [16]:
```python
# Bài 1.5: Viết lớp Rectangle có phương thức tính diện tích và chu vi.
class Rectangle:
    def __init__(self, height, width):
        self.height = height
        self.width = width
    def area (self):
        return self.height * self.width
    def perimeter(self):
        return 2* (self.height + self.width)

#test case
r1 = Rectangle(2,2)
print(f"Height is 2 and width is 2: area = {r1.area()}, perimeter = {r1.perimete
```

```
Height is 2 and width is 2: area = 4, perimeter = 8
```

In [18]:
```python
# Bài 1.6: Tạo lớp Book với các thuộc tính title, author, price. Viết phương
# thức giảm giá 10%
class Book:
    def __init__(self, title, author, price):
        self.title = title
        self.author = author
        self.price = price

    def discount(self):
        self.price = self.price*0.9
        return self.price

# test
b1 = Book ("Programming for beginer", "Alex", 100)
b1.discount()
```

Out[18]: 90.0

In [22]:
```python
# Bài 1.7: Xây dựng lớp Temperature có phương thức chuyển đổi từ Celsius sang Fa
class Temperature:
    def to_Fahrenheit(self, celsius):
        return celsius * 9/5 + 32

t1 = Temperature()
print(t1.to_Fahrenheit(32))
```

```
89.6
```

In [28]:
```python
# Bài 1.8: Tạo lớp Calculator thực hiện các phép tính cộng, trừ, nhân, chia.
class Calculator:
    def add(self, a, b):
        return a+ b
    def subtract(self, a, b):
        return a-b
    def multiply(self, a, b):
        return a*b
    def divide(self, a, b):
        if b == 0:
            return "Can't devided to zero"
        return a/b

print(f"a = 1 and b = 2\n")
a = 1
b= 2
```

```
c1 = Calculator()
print("a+b=", c1.add(a, b))
print("a-b=", c1.subtract(a, b))
print("a*b=", c1.multiply(a, b))
print("a/b=", c1.divide(a, b))
```

```
a = 1 and b = 2

a+b= 3
a-b= -1
a*b= 2
a/b= 0.5
```

In [30]:
```
# Bài 1.9: Tạo lớp Point biểu diễn toạ độ (x, y) và tính khoảng cách đến gốc toạ

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def distance(self):
        return math.sqrt(self.x**2 + self.y**2)

p1 = Point(3, 4)
print(p1.distance())
```

```
5.0
```

In [3]:
```
# Bài 1.10: Tạo lớp ComplexNumber mô phỏng số phức với các phép cộng, trừ
class ComplexNumber:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag
    def add(self, other):
        return ComplexNumber(self.real + other.real, self.imag + other.imag)
    def substract(self, other):
        return ComplexNumber(self.real - other.real, self.imag - other.imag)
    def __str__(self):
        return f"{self.real} + {self.imag}i"

c1 = ComplexNumber(2, 1)
c2 = ComplexNumber(1, -2)
result = c1.add(c2)
print(f"c1 = {c1}")
print(f"c2 = {c2}")
print(f"c1 + c2 = {result}")
```

```
c1 = 2 + 1i
c2 = 1 + -2i
c1 + c2 = 3 + -1i
```

Giai đoạn 2: Thuộc tính và phương thức nâng cao (10 bài)

In [4]:
```
# 2.11: Viết lớp Employee với thuộc tính lớp company = "TechCorp" và thuộc tính
# tượng name, salary
class Employee:
    company = "TechCorp"
    def __init__(self, name, salary):
```

```
        self.name = name
        self.salary = salary

    def info(self):
        print(f"Company: {Employee.company}, Name: {self.name}, Salary: {self.sa

e1 = Employee("THACO", 1000)
e1.info()
```

```
Company: TechCorp, Name: THACO, Salary: 1000
```

In [5]:
```
# Bài 2.12 + 2.13:
class Employee:
    company = "TechCorp"
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
    def info(self):
        print(f"Company: {Employee.company}, Name: {self.name}, Salary: {self.sa
    @classmethod
    def charge_company(cls, new_name):
        cls.company = new_name
    @staticmethod
    def is_valid_name(name):
        return isinstance(name, str) and len(name.strip())

e1 = Employee("Alex", 1000)
e2 = Employee("Bill", 2000)
e1.info()
e2.info()
Employee.charge_company('FB')
e1.info()
e2.info()
print(Employee.is_valid_name('Charlie'))
```

```
Company: TechCorp, Name: Alex, Salary: 1000
Company: TechCorp, Name: Bill, Salary: 2000
Company: FB, Name: Alex, Salary: 1000
Company: FB, Name: Bill, Salary: 2000
7
```

In [6]:
```
# Bài 2.14: Viết lớp BankAccount với thuộc tính riêng __balance, phương thức dep
# withdraw() có kiểm tra hợp lệ.
class BankAcount:
    def __init__(self, balance):
        self.__balance = balance
    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
        print (f"Deposit: {amount}. Balance: {self.__balance}")
    def withdraw(self, amount):
        if 0 < amount & amount < self.__balance:
            self.__balance -= amount
        print(f"Withdraw: {amount}, Balance: {self.__balance}")

    def get_balance(self):
        return self.__balance
ba1 = BankAcount(10000)
print(f"Balance: {ba1.get_balance()}")
```

```
ba1.deposit(2000)
ba1.withdraw(5000)
```

```
Balance: 10000
Deposit: 2000. Balance: 12000
Withdraw: 5000, Balance: 7000
```

In [7]:
```
# Bài 2.15: Tạo lớp Counter đếm số đối tượng được tạo ra
class Counter:
    count = 0
    def __init__(self):
        Counter.count += 1
    @classmethod
    def get_count(cls):
        return cls.count

c1 = Counter()
c2 = Counter()
c3 = Counter()
print(f"Total objects created: {Counter.get_count()}")
```

```
Total objects created: 3
```

In [8]:
```
# Bài 2.16: Viết lớp Product có thuộc tính ẩn __price và phương thức set_price()
class Product:
    def __init__(self, name, price):
        self.name = name
        self.__price = price

    def set_price(self, price):
        if price > 0:
            self.__price = price
            print(f"Price set to {price}")
        else:
            print("Price must be greater than 0")

    def get_price(self):
        return self.__price

p1 = Product("Laptop", 1000)
p1.set_price(1500)
p1.set_price(-100)
```

```
Price set to 1500
Price must be greater than 0
```

In [9]:
```
# Bài 2.17: Xây dựng lớp Vector2D có các phép cộng, trừ vector.
class Vector2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def add(self, other):
        return Vector2D(self.x + other.x, self.y + other.y)

    def subtract(self, other):
        return Vector2D(self.x - other.x, self.y - other.y)

    def __str__(self):
        return f"({self.x}, {self.y})"
```

```
v1 = Vector2D(1, 2)
v2 = Vector2D(3, 4)
v3 = v1.add(v2)
v4 = v1.subtract(v2)
print(f"v1 = {v1}, v2 = {v2}")
print(f"v1 + v2 = {v3}")
print(f"v1 - v2 = {v4}")
```

```
v1 = (1, 2), v2 = (3, 4)
v1 + v2 = (4, 6)
v1 - v2 = (-2, -2)
```

In [10]:
```
# Bài 2.18: Viết lớp Polynomial biểu diễn đa thức, hỗ trợ in ra dạng anxn+···+a0
class Polynomial:
    def __init__(self, coefficients):
        self.coefficients = coefficients

    def __str__(self):
        terms = []
        n = len(self.coefficients) - 1
        for i, coeff in enumerate(self.coefficients):
            power = n - i
            if coeff == 0:
                continue
            if power == 0:
                terms.append(f"{coeff}")
            elif power == 1:
                terms.append(f"{coeff}x")
            else:
                terms.append(f"{coeff}x^{power}")
        return " + ".join(terms)

    def evaluate(self, x):
        result = 0
        for i, coeff in enumerate(self.coefficients):
            power = len(self.coefficients) - 1 - i
            result += coeff * (x ** power)
        return result

p = Polynomial([1, -5, 6])
print(f"Polynomial: {p}")
print(f"p(2) = {p.evaluate(2)}")
```

```
Polynomial: 1x^2 + -5x + 6
p(2) = 0
```

In [11]:
```
# Bài 2.19: Viết lớp Logger ghi lại các hoạt động của người dùng vào file.
import datetime

class Logger:
    def __init__(self, filename):
        self.filename = filename

    def log(self, message):
        with open(self.filename, 'a') as f:
            timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            f.write(f"[{timestamp}] {message}\n")

    def read_logs(self):
        with open(self.filename, 'r') as f:
            return f.read()
```

```
logger = Logger("app.log")
logger.log("User logged in")
logger.log("User performed action X")
print(logger.read_logs())
```

```
[2025-11-16 09:17:12] User logged in
[2025-11-16 09:17:12] User performed action X
```

In [12]:
```
# Bài 2.20: Viết lớp Timer có phương thức start(), stop(), và elapsed() để đo th
import time

class Timer:
    def __init__(self):
        self.start_time = None
        self.end_time = None

    def start(self):
        self.start_time = time.time()
        print("Timer started")

    def stop(self):
        self.end_time = time.time()
        print("Timer stopped")

    def elapsed(self):
        if self.start_time is None or self.end_time is None:
            return "Timer not properly used"
        return self.end_time - self.start_time

timer = Timer()
timer.start()
time.sleep(2)
timer.stop()
print(f"Elapsed time: {timer.elapsed():.2f} seconds")
```

```
Timer started
Timer stopped
Elapsed time: 2.00 seconds
Timer stopped
Elapsed time: 2.00 seconds
```

Giai đoạn 3: Kế thừa và Đa hình

In [13]:
```
# Bài 3.21: Tạo lớp cha Person và lớp con Student(Person) kế thừa các thuộc tính
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display(self):
        print(f"Name: {self.name}, Age: {self.age}")

class Student(Person):
    def __init__(self, name, age, student_id):
        super().__init__(name, age)
        self.student_id = student_id
```

```
s1 = Student("An", 20, "001")
s1.display()
print(f"Student ID: {s1.student_id}")
```

```
Name: An, Age: 20
Student ID: 001
```

In [14]:
```
# Bài 3.22: Viết lớp Teacher(Person) có thêm thuộc tính subject.
class Teacher(Person):
    def __init__(self, name, age, subject):
        super().__init__(name, age)
        self.subject = subject

    def display(self):
        super().display()
        print(f"Subject: {self.subject}")

t1 = Teacher("Mr. John", 40, "Mathematics")
t1.display()
```

```
Name: Mr. John, Age: 40
Subject: Mathematics
```

In [15]:
```
# Bài 3.23: Tạo lớp ResearchStudent(Student) có thêm topic.
class ResearchStudent(Student):
    def __init__(self, name, age, student_id, topic):
        super().__init__(name, age, student_id)
        self.topic = topic

    def display(self):
        super().display()
        print(f"Topic: {self.topic}")

rs1 = ResearchStudent("Bình", 21, "002", "Machine Learning")
rs1.display()
```

```
Name: Bình, Age: 21
Topic: Machine Learning
```

In [16]:
```
# Bài 3.24: Ghi đè phương thức (method overriding) trong lớp con để hiển thị thô
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def display(self):
        print(f"Employee: {self.name}, Salary: {self.salary}")

class Manager(Employee):
    def __init__(self, name, salary, department):
        super().__init__(name, salary)
        self.department = department

    def display(self):
        super().display()
        print(f"Department: {self.department}")

m1 = Manager("Alice", 5000, "IT")
m1.display()
```

```
Employee: Alice, Salary: 5000
Department: IT
```

In [17]:
```python
# Bài 3.25: Tạo lớp Animal, các lớp con Dog, Cat có phương thức speak() thể hiện
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print(f"{self.name} makes a sound")

class Dog(Animal):
    def speak(self):
        print(f"{self.name} barks: Woof! Woof!")

class Cat(Animal):
    def speak(self):
        print(f"{self.name} meows: Meow! Meow!")

d1 = Dog("Buddy")
c1 = Cat("Whiskers")
d1.speak()
c1.speak()
```

```
Buddy barks: Woof! Woof!
Whiskers meows: Meow! Meow!
```

In [18]:
```python
# Bài 3.26: Xây dựng lớp Shape trừu tượng và các lớp con Rectangle, Circle, Tria
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        import math
        return math.pi * self.radius ** 2

class Triangle(Shape):
    def __init__(self, base, height):
        self.base = base
        self.height = height

    def area(self):
        return 0.5 * self.base * self.height

r1 = Rectangle(5, 4)
```

```
c1 = Circle(3)
t1 = Triangle(6, 4)
print(f"Rectangle area: {r1.area()}")
print(f"Circle area: {c1.area():.2f}")
print(f"Triangle area: {t1.area()}")
```

```
Rectangle area: 20
Circle area: 28.27
Triangle area: 12.0
```

In [19]:
```
# Bài 3.27: Viết hàm nhận danh sách các Shape và tính tổng diện tích (đa hình).
def total_area(shapes):
    total = 0
    for shape in shapes:
        total += shape.area()
    return total

shapes = [Rectangle(5, 4), Circle(3), Triangle(6, 4)]
print(f"Total area: {total_area(shapes):.2f}")
```

```
Total area: 60.27
```

In [20]:
```
# Bài 3.28: Xây dựng hệ thống quản lý nhân sự gồm lớp cha Employee và các lớp co
class BaseEmployee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def display(self):
        print(f"Name: {self.name}, Salary: {self.salary}")

class ManagerEmp(BaseEmployee):
    def __init__(self, name, salary, department):
        super().__init__(name, salary)
        self.department = department

    def display(self):
        super().display()
        print(f"Position: Manager, Department: {self.department}")

class Engineer(BaseEmployee):
    def __init__(self, name, salary, language):
        super().__init__(name, salary)
        self.language = language

    def display(self):
        super().display()
        print(f"Position: Engineer, Language: {self.language}")

class Intern(BaseEmployee):
    def __init__(self, name, salary, university):
        super().__init__(name, salary)
        self.university = university

    def display(self):
        super().display()
        print(f"Position: Intern, University: {self.university}")

m = ManagerEmp("David", 5000, "IT")
e = Engineer("Emily", 3500, "Python")
i = Intern("Frank", 1000, "MIT")
```

```
m.display()
e.display()
i.display()
```

```
Name: David, Salary: 5000
Position: Manager, Department: IT
Name: Emily, Salary: 3500
Position: Engineer, Language: Python
Name: Frank, Salary: 1000
Position: Intern, University: MIT
```

In [21]:
```
# Bài 3.29: Viết ví dụ kế thừa đa tầng: Vehicle → Car → ElectricCar.
class Vehicle:
    def __init__(self, brand):
        self.brand = brand

    def display(self):
        print(f"Brand: {self.brand}")

class Car(Vehicle):
    def __init__(self, brand, num_doors):
        super().__init__(brand)
        self.num_doors = num_doors

    def display(self):
        super().display()
        print(f"Type: Car, Doors: {self.num_doors}")

class ElectricCar(Car):
    def __init__(self, brand, num_doors, battery):
        super().__init__(brand, num_doors)
        self.battery = battery

    def display(self):
        super().display()
        print(f"Battery: {self.battery}kWh")

ec = ElectricCar("Tesla", 4, 100)
ec.display()
```

```
Brand: Tesla
Type: Car, Doors: 4
Battery: 100kWh
```

In [22]:
```
# Bài 3.30: Tạo ví dụ kế thừa đa kế (multiple inheritance) giữa lớp Flyable và S
class Flyable:
    def fly(self):
        print("This creature can fly")

class Swimmable:
    def swim(self):
        print("This creature can swim")

class Duck(Flyable, Swimmable):
    def quack(self):
        print("Quack! Quack!")

d = Duck()
d.fly()
```

```
d.swim()
d.quack()
```

```
This creature can fly
This creature can swim
Quack! Quack!
```

Giai đoạn 4: Ứng dụng OOP trong Khoa học Dữ liệu

In [35]:
```
# Bài 4.31: Viết lớp DataLoader đọc file CSV bằng pandas, có phương thức load()
import pandas as pd

class DataLoader:
    def __init__(self, filepath=None, data=None):
        self.filepath = filepath
        self.data = data

    def load(self):
        if self.filepath:
            self.data = pd.read_csv(self.filepath)
            print(f"Data loaded from {self.filepath}")
        return self.data

    def summary(self):
        if self.data is None:
            print("Data not loaded yet")
            return
        print(self.data.describe())
        print(f"\nShape: {self.data.shape}")

sample_df = pd.DataFrame({
    'feature1': [1.2, 2.5, 1.8, 3.2, 2.1, 4.5, 2.8, 3.9, 1.5, 3.3],
    'feature2': [10, 25, 18, 32, 21, 45, 28, 39, 15, 33],
    'target': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
})
loader = DataLoader(data=sample_df)
loader.load()
loader.summary()
```

```
       feature1   feature2     target
count  10.000000  10.000000  10.000000
mean    2.680000  26.600000   0.500000
std     1.064373  10.966616   0.527046
min     1.200000  10.000000   0.000000
25%     1.875000  18.750000   0.000000
50%     2.650000  26.500000   0.500000
75%     3.275000  32.750000   1.000000
max     4.500000  45.000000   1.000000

Shape: (10, 3)
```

In [36]:
```
# Bài 4.32: Tạo lớp Preprocessor có phương thức xử lý giá trị thiếu và chuẩn hóa
import numpy as np

class Preprocessor:
    def __init__(self, data):
        self.data = data.copy()
```

```
    def handle_missing_values(self, method='mean'):
        if method == 'mean':
            self.data.fillna(self.data.mean(), inplace=True)
        elif method == 'median':
            self.data.fillna(self.data.median(), inplace=True)
        elif method == 'drop':
            self.data.dropna(inplace=True)
        return self.data

    def normalize(self):
        return (self.data - self.data.min()) / (self.data.max() - self.data.min(

    def standardize(self):
        return (self.data - self.data.mean()) / self.data.std()

sample_data = pd.DataFrame({
    'A': [1.2, 2.5, np.nan, 3.2, 2.1, 4.5, np.nan, 3.9, 1.5, 3.3],
    'B': [10, 25, 18, np.nan, 21, 45, 28, 39, np.nan, 33]
})
preprocessor = Preprocessor(sample_data)
print("Original data:")
print(preprocessor.data)
print("\nAfter handling missing values:")
print(preprocessor.handle_missing_values())
```

```
Original data:
     A     B
0  1.2  10.0
1  2.5  25.0
2  NaN  18.0
3  3.2   NaN
4  2.1  21.0
5  4.5  45.0
6  NaN  28.0
7  3.9  39.0
8  1.5   NaN
9  3.3  33.0

After handling missing values:
       A       B
0  1.200  10.000
1  2.500  25.000
2  2.775  18.000
3  3.200  27.375
4  2.100  21.000
5  4.500  45.000
6  2.775  28.000
7  3.900  39.000
8  1.500  27.375
9  3.300  33.000
```

In [37]:
```
# Bài 4.33: Xây dựng lớp DatasetSplitter chia dữ liệu thành train/test.
from sklearn.model_selection import train_test_split

class DatasetSplitter:
    def __init__(self, data, test_size=0.2, random_state=42):
        self.data = data
        self.test_size = test_size
        self.random_state = random_state
        self.X_train = None
```

```
        self.X_test = None
        self.y_train = None
        self.y_test = None

    def split(self, features, target):
        X = self.data[features]
        y = self.data[target]
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
            X, y, test_size=self.test_size, random_state=self.random_state
        )
        print(f"Train set: {self.X_train.shape}, Test set: {self.X_test.shape}")
        return self.X_train, self.X_test, self.y_train, self.y_test

df = pd.DataFrame({
    'feature1': [1.2, 2.5, 1.8, 3.2, 2.1, 4.5, 2.8, 3.9, 1.5, 3.3],
    'feature2': [10, 25, 18, 32, 21, 45, 28, 39, 15, 33],
    'target': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
})
splitter = DatasetSplitter(df)
X_train, X_test, y_train, y_test = splitter.split(['feature1', 'feature2'], 'tar
```

```
Train set: (8, 2), Test set: (2, 2)
```

In [38]:

```
# Bài 4.34: Viết lớp StatisticsAnalyzer có các phương thức tính mean(), variance
class StatisticsAnalyzer:
    def __init__(self, data):
        self.data = data

    def mean(self, column=None):
        if column:
            return self.data[column].mean()
        return self.data.mean()

    def variance(self, column=None):
        if column:
            return self.data[column].var()
        return self.data.var()

    def correlation(self):
        return self.data.corr()

    def std(self, column=None):
        if column:
            return self.data[column].std()
        return self.data.std()

analyzer = StatisticsAnalyzer(df)
print(f"Mean: {analyzer.mean()}")
print(f"\nVariance: {analyzer.variance()}")
print(f"\nCorrelation:\n{analyzer.correlation()}")
```

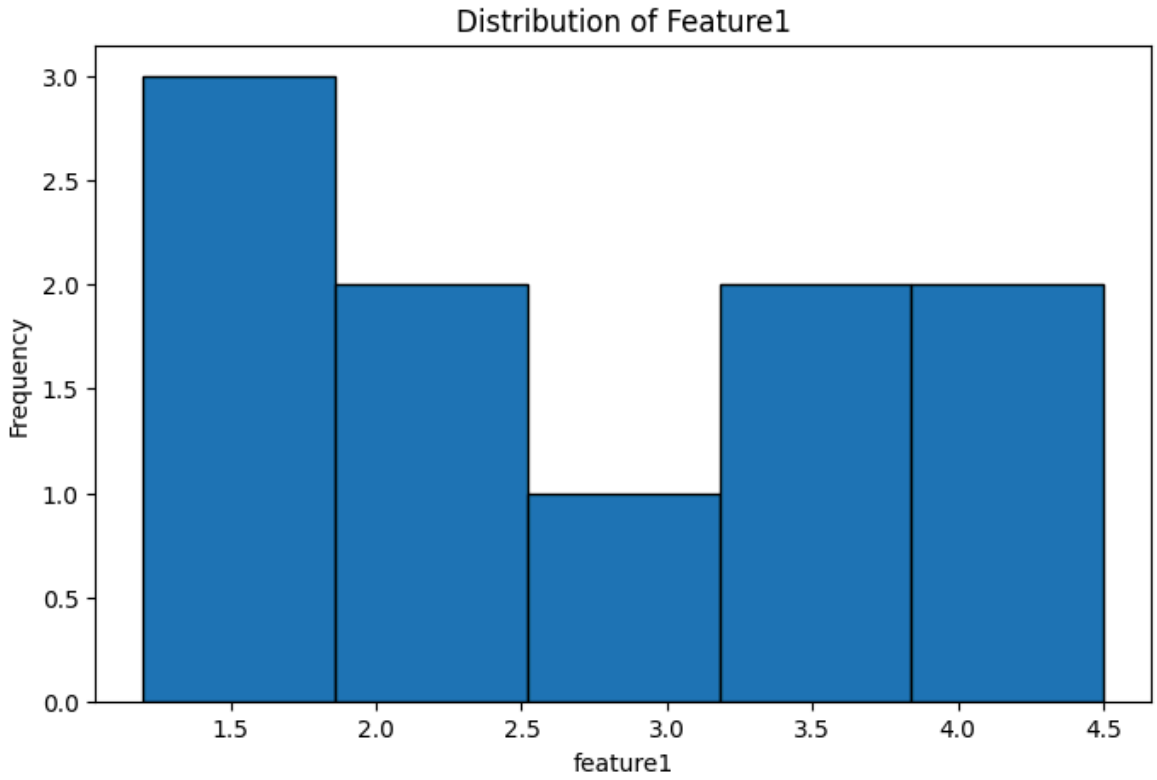
```
Mean: feature1     2.68
feature2    26.60
target       0.50
dtype: float64

Variance: feature1      1.132889
feature2    120.266667
target        0.277778
dtype: float64

Correlation:
          feature1  feature2    target
feature1  1.000000  0.998733  0.792273
feature2  0.998733  1.000000  0.788170
target    0.792273  0.788170  1.000000
```

In [39]:
```python
# Bài 4.35: Viết lớp FeatureSelector chọn cột dữ liệu theo điều kiện.
class FeatureSelector:
    def __init__(self, data):
        self.data = data
        self.selected_features = []

    def select_by_type(self, dtype):
        self.selected_features = self.data.select_dtypes(include=[dtype]).column
        return self.selected_features

    def select_by_variance(self, threshold=0.5):
        variances = self.data.var()
        self.selected_features = variances[variances > threshold].index.tolist()
        return self.selected_features

    def get_selected_data(self):
        return self.data[self.selected_features]

selector = FeatureSelector(df)
print(f"Numeric features: {selector.select_by_type('number')}")
print(f"Selected data:\n{selector.get_selected_data()}")
```

```
Numeric features: ['feature1', 'feature2', 'target']
Selected data:
   feature1  feature2  target
0       1.2        10       0
1       2.5        25       1
2       1.8        18       0
3       3.2        32       1
4       2.1        21       0
5       4.5        45       1
6       2.8        28       0
7       3.9        39       1
8       1.5        15       0
9       3.3        33       1
```

In [40]:
```python
# Bài 4.36: Tạo lớp Visualizer vẽ biểu đồ histogram, scatter bằng matplotlib.
import matplotlib.pyplot as plt

class Visualizer:
    def __init__(self, data):
        self.data = data

    def histogram(self, column, bins=10, title=None):
```

```
        plt.figure(figsize=(8, 5))
        plt.hist(self.data[column], bins=bins, edgecolor='black')
        plt.title(title or f"Histogram of {column}")
        plt.xlabel(column)
        plt.ylabel("Frequency")
        plt.show()

    def scatter(self, x_col, y_col, title=None):
        plt.figure(figsize=(8, 5))
        plt.scatter(self.data[x_col], self.data[y_col])
        plt.title(title or f"Scatter plot: {x_col} vs {y_col}")
        plt.xlabel(x_col)
        plt.ylabel(y_col)
        plt.show()

visualizer = Visualizer(df)
visualizer.histogram('feature1', bins=5, title="Distribution of Feature1")
```



In [41]:
```
# Bài 4.37: Viết lớp LinearModel với fit(), predict(), evaluate().
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

class LinearModel:
    def __init__(self):
        self.model = LinearRegression()
        self.is_trained = False

    def fit(self, X_train, y_train):
        self.model.fit(X_train, y_train)
        self.is_trained = True
        print("Model fitted")

    def predict(self, X):
        if not self.is_trained:
            print("Model not trained yet")
            return None
```

```
        return self.model.predict(X)

    def evaluate(self, X_test, y_test):
        if not self.is_trained:
            print("Model not trained yet")
            return None
        predictions = self.predict(X_test)
        mse = mean_squared_error(y_test, predictions)
        r2 = r2_score(y_test, predictions)
        print(f"MSE: {mse:.4f}, R2: {r2:.4f}")
        return mse, r2

model = LinearModel()
model.fit(X_train, y_train)
model.evaluate(X_test, y_test)
```

```
Model fitted
MSE: 0.2635, R2: -0.0542
```

Out[41]: (0.263546329639889, -0.054185318559555995)

In [42]:
```
# Bài 4.38: Tạo lớp Scaler thực hiện chuẩn hóa MinMax hoặc StandardScaler.
from sklearn.preprocessing import MinMaxScaler, StandardScaler

class Scaler:
    def __init__(self, method='minmax'):
        self.method = method
        if method == 'minmax':
            self.scaler = MinMaxScaler()
        elif method == 'standard':
            self.scaler = StandardScaler()
        else:
            raise ValueError("Method must be 'minmax' or 'standard'")

    def fit_transform(self, X):
        return self.scaler.fit_transform(X)

    def transform(self, X):
        return self.scaler.transform(X)

scaler = Scaler(method='minmax')
scaled_data = scaler.fit_transform(X_train)
print(f"Original shape: {X_train.shape}, Scaled shape: {scaled_data.shape}")
```

```
Original shape: (8, 2), Scaled shape: (8, 2)
```

In [43]:
```
# Bài 4.39: Xây dựng lớp Pipeline kết hợp các bước tiền xử lý và huấn luyện mô h
class Pipeline:
    def __init__(self):
        self.steps = []

    def add_step(self, name, transformer):
        self.steps.append((name, transformer))

    def fit_transform(self, X, y=None):
        for name, transformer in self.steps:
            if hasattr(transformer, 'fit_transform'):
                X = transformer.fit_transform(X)
            elif hasattr(transformer, 'fit'):
                transformer.fit(X, y)
                if hasattr(transformer, 'transform'):
```

```
                    X = transformer.transform(X)
        return X

    def fit(self, X, y=None):
        for name, transformer in self.steps[:-1]:
            if hasattr(transformer, 'fit_transform'):
                X = transformer.fit_transform(X)
            else:
                transformer.fit(X, y)
        self.steps[-1][1].fit(X, y)

pipeline = Pipeline()
pipeline.add_step('scaler', Scaler('minmax'))
pipeline.add_step('model', LinearModel())
print("Pipeline created with scaler and model")
```

```
Pipeline created with scaler and model
```

In [44]:

```
# Bài 4.40: Viết lớp ReportGenerator sinh báo cáo tóm tắt kết quả mô hình ra fil
class ReportGenerator:
    def __init__(self, model_name, dataset_name):
        self.model_name = model_name
        self.dataset_name = dataset_name
        self.report = ""

    def add_section(self, title, content):
        self.report += f"\n{'='*50}\n"
        self.report += f"{title}\n"
        self.report += f"{'='*50}\n"
        self.report += f"{content}\n"

    def generate_report(self, train_score, test_score, metrics):
        self.add_section("MODEL REPORT", f"Model: {self.model_name}\nDataset: {s
        self.add_section("TRAINING RESULTS", f"Train Score: {train_score:.4f}\nT
        self.add_section("METRICS", metrics)
        return self.report

    def save_report(self, filename):
        with open(filename, 'w') as f:
            f.write(self.report)
        print(f"Report saved to {filename}")

report_gen = ReportGenerator("Linear Regression", "Sample Dataset")
report = report_gen.generate_report(0.85, 0.82, "MSE: 0.5, R2: 0.82")
report_gen.save_report("model_report.txt")
print(report)
```

```
Report saved to model_report.txt

==================================================
MODEL REPORT
==================================================
Model: Linear Regression
Dataset: Sample Dataset

==================================================
TRAINING RESULTS
==================================================
Train Score: 0.8500
Test Score: 0.8200

==================================================
METRICS
==================================================
MSE: 0.5, R2: 0.82
```