COURSE WORK: PROGRAMMING FOR DATA SCIENCE 2025.2 BATCH

MSc Data Science: Coventry University UK

Submitted by: M.L.S. Weerasingha

Student Number: COMScDS252P_014

Contents

1 Executive Summary

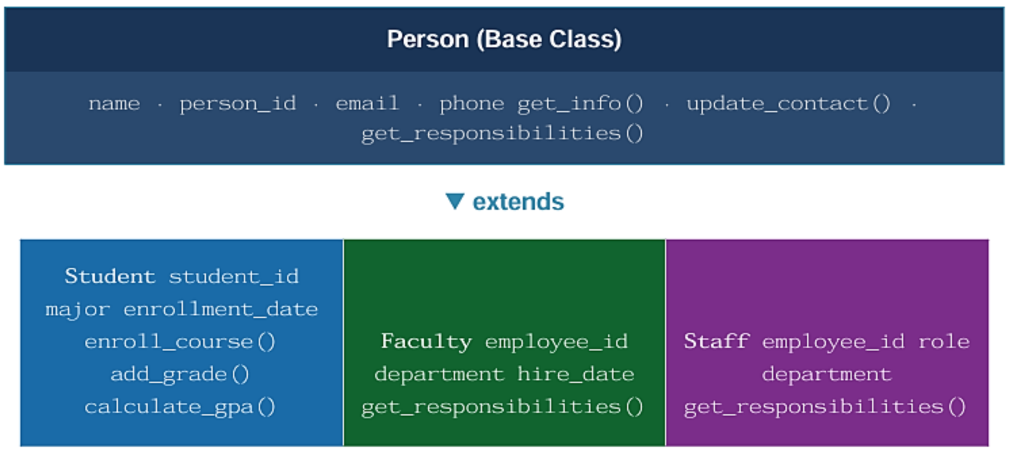
This is a three-part technical coursework report of Object-Oriented Programming, Data Analysis, and AI Ethics. Built an University Management System with Full OOP features using Python 3.14.0 We scraped, cleaned and analysed 200 books from books using an e-commerce data pipeline. toscrape. com and found no relationship between price and rating. These findings were contextualized within an ethical framework with respect to AI in healthcare, analyzing three known challenges—algorithmic bias and privacy risks—as well as several broad recommendations relevant for clinical use of AI products.

2 Question 1: OOP Implementation

2.1 Architecture and Class Hierarchy

The University Management System (UMS) is built on a two level inheritance system and is based on a Person class that contains all the shared identification fields of the people from whom data is collected (name, person_id, email and telephone) as well as the two common methods (get info(), for formatting contact information, and update contact(), for modifying contact information) associated with all members of the general identification/record system hierarchy.

Three specialized subclasses extend Person directly:


Person (Base Class)
name · person_id · email · phone get_info() · update_contact() · get_responsibilities()
▼ extends
Student student_id major enrollment_date enroll_course() add_grade() calculate_gpa()
Faculty employee_id department hire_date get_responsibilities()
Staff employee_id role department get_responsibilities()

2.2 Design decisions

Several deliberate design choices were made to maximise cohesion and minimise coupling.

- **GPA with Read-Only Access:**
  GPA is a read-only property computed dynamically using the live grades dictionary and will always have a consistent value. Name Mangled and Controlled
- **Grade Storage:**
  The internal grades dictionary (_grades) is name-mangled for protection from external modification and can only be accessed via a getter method that provides a shallow copy of the grades for viewing without the ability to change them.
- **Single enrollments point of truth:**
  Course.add_student() is the one and only source of truth when it comes to adding a student into the course. It verifies that the course has the capacity to accept an additional student, whether there are and exists duplicate students being added to the course and

then calls the Student.enroll_course() function to keep both sides of the enrollment relationship in sync without requiring the caller to keep track of either object (course or student).

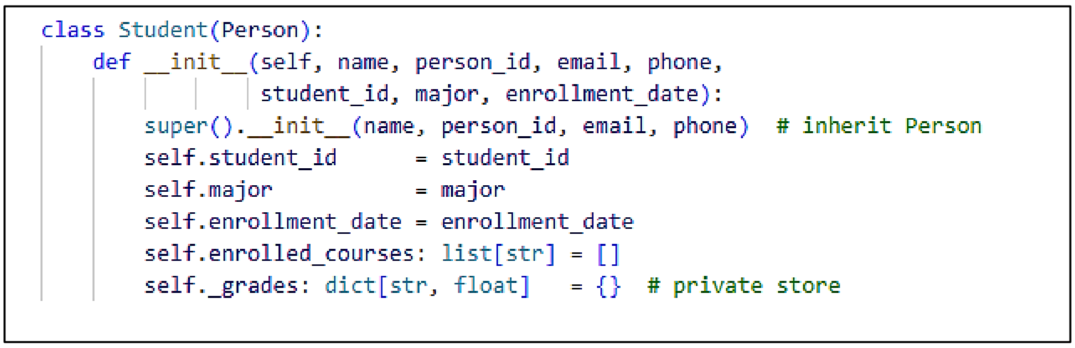
- **Automatic registration of Department Head:**
  The init method for the Department class automatically registers the department head in the faculty list, which helps to avoid a department head belonging to a department that doesn't exist in the list of faculty members.
- **Defensive Error Handling:**
  All validation will raise ValueError with a clear description of what went wrong, and the demonstration script wraps every action in a safe_action method that catches and logs errors neatly, simulating a real-life use of defensive programming.

2.3 Key OOP Features Demonstrated

Feature	Where Applied	Benefit
Inheritance	Student / Faculty / Staff ← Person	Shared interface, code reuse
Encapsulation	_grades (private), @property gpa	Data integrity, read-only GPA
Polymorphism	get_responsibilities() overridden ×3	Heterogeneous list processing
Abstraction	get_info() / get_responsibilities()	Uniform API across all types
Composition	Department holds Faculty & Course lists	Flexible domain modelling
Validation	add_grade(): 0.0–4.0; max 6 courses	Prevents corrupt state

2.4 Selected Code Snippets with Explanations

2.4.1 Snippet 1 — Inheritance via super()



```
class Student(Person):
    def __init__(self, name, person_id, email, phone,
                 student_id, major, enrollment_date):
        super().__init__(name, person_id, email, phone)  # inherit Person
        self.student_id      = student_id
        self.major           = major
        self.enrollment_date = enrollment_date
        self.enrolled_courses: list[str] = []
        self._grades: dict[str, float]   = {}  # private store
```

When the Student constructor calls super().__init__(), it initializes four Person attributes as well as its own attributes. By doing this, we are ensuring Method Resolution Order is honoured and we avoid duplicate attributes in the hierarchy.

2.4.2 Snippet 2 — Encapsulation & Validation

```
# student.py — gpa property + add_grade validation
@property
def gpa(self) -> float:
    """Read-only: always re-calculated from live grades."""
    if not self._grades:
        return 0.0
    return sum(self._grades.values()) / len(self._grades)

def add_grade(self, course_code: str, grade: float) -> None:
    if not (0.0 <= float(grade) <= 4.0):
        raise ValueError("Grade must be between 0.0 and 4.0.")
    self._grades[course_code.strip().upper()] = float(grade)
```

The GPA property is computed on demand from the list of grades as indicated by the presence of an empty setter, which makes it purely read-only. The add_grade() method validates that any added grades fall between 0.0 and 4.0 and raises a ValueError if a violation occurs, thereby safeguarding the model from being placed into an invalid position.

2.4.3 Snippet 3 — Polymorphism

```
# main.py — polymorphism demonstration
people = [stu1, fac1, staff1]   # mixed types in one list
for p in people:
    print(f"{p.name}: {p.get_responsibilities()}")

# Output:
# Lahiru Weerasingha: Attend lectures, complete coursework ...
# Dr. Silva: Deliver lectures, supervise students ...
# Ms. Nuwan: Provide administrative support ...
```

A list of differing types (Student, Faculty and Staff) is iterated using a single for loop, and Python's dynamic dispatch mechanism calls the correct get_responsibilities() overridden method, demonstrated true polymorphic behavior at runtime, without any need to perform any isinstance checking.

3 Question 2: Data Analysis

3.1 Methodology & Tools

This study utilized a five-stage pipeline of systematic analysis to investigate e-commerce book data imported from **books.toscrape.com**. Each stage of analysis entailed the integration of the following tools via at least one of the five stages listed below: Raw Data Acquisition, Data Cleaning, Inferential Statistics, Data Visualization, and Predictive Modelling. The final form of this pipeline is built upon the following software packages; **requests; BeautifulSoup4; pandas; NumPy; SciPy; Matplotlib; Plotly; Scikit-learn** utilizing **Python 3.14.0**. In addition, all downloaded data was expected to have a one to two second randomized delay between each HTTP request to ensure compliance with server rate limits. Finally, when a network failure occurred, each data set was re-downloaded a maximum of 3 times before stopping the attempt.

Stage	Script	Key Output
Web Scraping	1_scraper.py	raw_books_data.csv (200 books)
Data Cleaning	2_data_cleaning.py	cleaned_books_data.csv + 2 derived cols
Statistical Analysis	3_analysis.py	Descriptive + inferential stats
Visualization	4_visualization.py	4 charts (2 static, 2 interactive)
Predictive Analysis	5_prediction.py	LinearRegression R^2 & MAE

3.2 Data Collection & Cleaning

The scraper extracted a total of **200 book records**: 20 books per each page from **10 pages of the catalogue**. The detail page for each book was then accessed, and the exact category of the book was extracted from the breadcrumb navigation in each detail page. As raw data was saved as a UTF-8-BOM CSV file, there is no risk of incorrectly rendering the pound sterling symbol across all operating systems.

The cleaning pipeline **indicated no missing or duplicate records** were present in the raw data, implying successful implementation of the scraper's error management capabilities. The price column has been standardized by **removing the £ symbol** from values so that they can be converted into **float64** type, whereas the rating column has been left in its original **int64 type**. In addition, two new features have been created: price_category (Budget: Less than £20, Mid-range: £20 to £40, Premium: More than £40) and in_stock (boolean). The final cleaned dataset contains 200 rows and 7 columns.

Metric	Before Cleaning	After Cleaning
Shape	(200, 5)	(200, 7)
Missing values	0	0
Duplicates	0	0
price_gbp type	str (raw £ string)	float64
rating type	int64	int64
Derived columns	—	price_category, in_stock

3.3 Statistical Results

3.3.1 Descriptive Statistics

Price (GBP):

Statistic	Value	Interpretation
Mean	£34.80	Books are moderately priced on average
Median	£35.64	Slightly above mean — slight left skew
Mode	£33.34	Most frequently occurring price point
Std Dev	£14.12	High spread — prices vary considerably
Range	£49.48 (£10.16–£59.64)	Wide price spectrum across categories

Distribution of Rating:

Rating	Frequency	Percentage of Total
1 ★	49	24.50%
2 ★★	41	20.50%
3 ★★★	35	17.50%
4 ★★★★	38	19.00%
5 ★★★★★	37	18.50%

3.3.2 Inferential Statistics

Test	Statistic	p-value	Conclusion
IQR Outliers	Zero outliers	—	No price outliers
Pearson r	r = 0.017	0.810	No significant correlation
t-test (Fic vs non-Fic)	t = 1.913	0.077	Fail to reject H_0

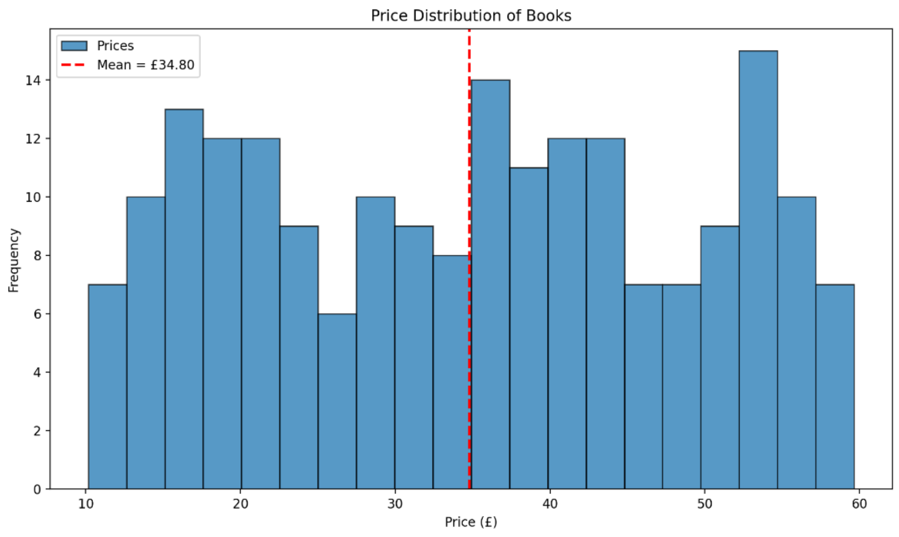
- Outlier Detection (IQR):
  Q1 = £21.99, Q3 = £46.11, IQR = £24.12, hence there are zero outlier prices for the books since all of them fall under these boundaries as maximum upper limit set at £82.29.
- Pearson Correlation (Price vs Rating):

r = 0.0171, p = 0.810 confirmed that there is no linear relationship between the price and rating. This finding is supported by both **a near-zero correlation coefficient** and a **non-significant p-value (α=0.05)** which indicate that price does not provide any insight regarding how highly a reader will rate a book.

- Independent t-test (Fiction vs Non-Fiction prices):
  t = 1.9133, p = 0.077. With p > 0.05; **fail to reject H_0**
  There is insufficient evidence to conclude that Fiction (n=9) and Non-Fiction (n=20) books are priced differently. The small Fiction sample size limits statistical power here.

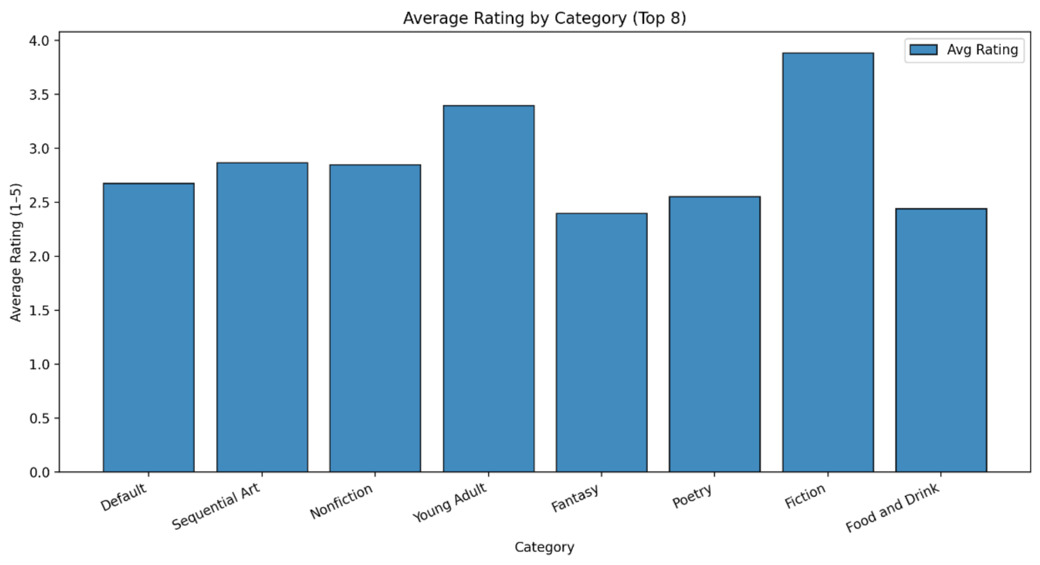
3.4 Visualizations

3.4.1 Figure 1 — Price Distribution Histogram



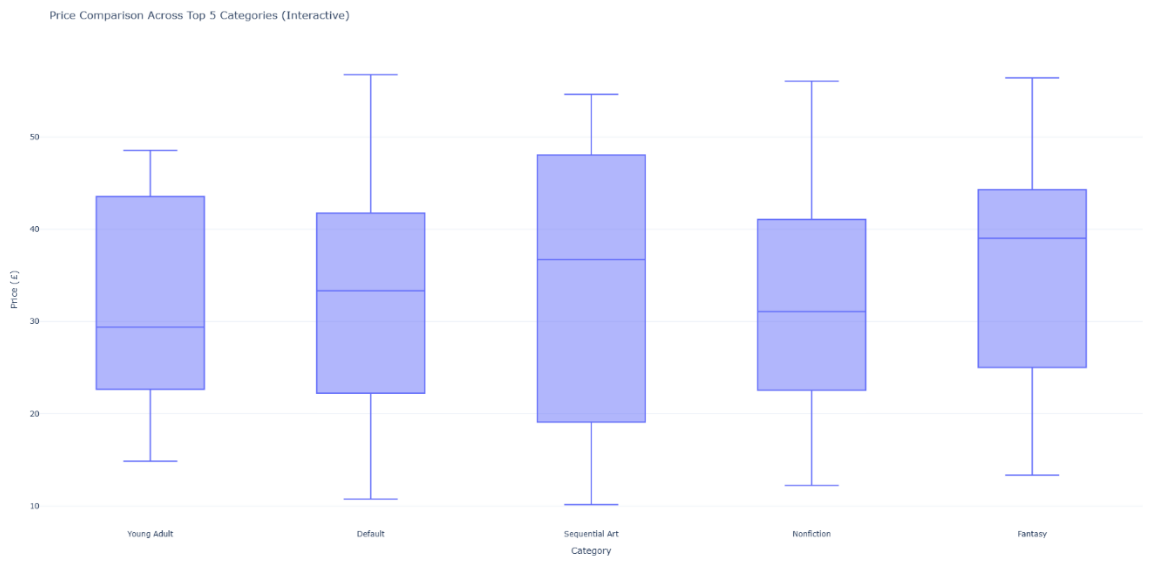
The histogram indicates that this data has two peaks where half of the books are priced between 14–22 and the other half at a higher price of 48–56. The average price is 34.80, and it is located between the two peaks of price. Therefore, we can see how relying only on the average as a measure can provide a false view of this dataset; in fact, the retailer who uses average price for product placement on the market will miss out on two separate customer groups who will potentially purchase books priced outside of the average range.

3.4.2 Figure 2 — Average Rating by Category (Top 8)



There is a clear variation in rating across categories. The mean for fiction is the highest at approximately 3.88, while the second-highest mean falls under young adults (approximately 3.40). Also, fantasy, and food & drink are at the bottom of the list of genres with averages around 2.40-2.45 each. This indicates that reader satisfaction varies by category, suggesting that genre selection significantly impacts perceived quality.

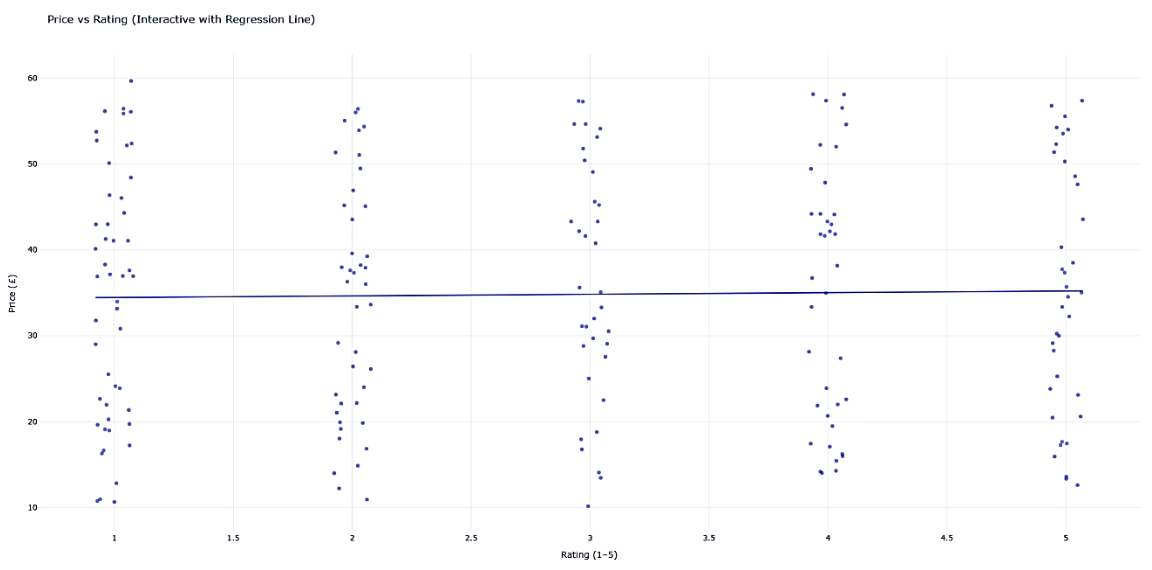
3.4.3 Figure 3 — Interactive Box Plot (Top 5 Categories)



The Price spread varies widely among the five most popular product categories. Sequential Art recorded the largest interquartile range (£19 - £48), showing significant variability in pricing.

The Nonfiction category had a narrower spread (£23 - £41). Additionally, all product categories have overlapping price ranges, as indicated by the t-test results not statistically significant.

3.4.4 Figure 4 — Price vs Rating Scatter Plot



The regression line shows a horizontal line suggesting that price and rating are independent variables regardless of genre or title.

3.5 Predictive Analysis

To predict Book Price as a Linear Regression implementation using rating and one-hot encoded category as features with 80/20 train-test data split and the random_state=42.

Metric	Value	Interpretation
R^2 Score	−0.1849	Model performs worse than a simple mean baseline
Mean Absolute Error	£14.07	Average prediction error of £14 per book
Strongest feature	Category	Category coefficients dominate over rating (0.42)

With a negative R^2 value, the linear regression using rating and genre as predictors does not yield a significant predictive signal, indicating that book pricing at books.toscrape.com does not strongly correlate to either ratings or genres independently when creating the prediction model. Although category dummies account for the largest weighted averages (e.g., Christian = £21.11, Autobiography = -£20.66), this reiterates that category is a better differentiator than rating for pricing but even in the case of categories, the levels of differentiation will not be reliable when trying to create a prediction model.

4 Question 3: Ethics Analysis

4.1 Framework Summary

The analysis uses a multi-stakeholder ethical framework to assess the use of AI in healthcare settings as they relate to Health Insurance Portability and Accountability Act (HIPAA, USA) and General Data Protection Regulation (GDPR, European Union). HIPAA requires administrative, physical, and technical safeguards for Protected Health Information (PHI) and that a formal risk assessment of how patient health data will be processed through an AI application is completed prior to allowing the AI application to process the data. GDPR classifies health data as a special category and requires explicit patient consent before any AI system can process the patient's health data; however, the requirement for a patient's Right to be Forgotten (Article 17) creates a conflict between patient rights and AI systems. A patient has the right to demand their data be deleted; however, the model's learned parameters associated with that deleted data cannot simply be deleted by the model.

4.2 Key Ethical Concerns

Foundational risks are posed by privacy issues in the healthcare field. An example of how simple de-identification is not sufficient, multiple studies have demonstrated that researchers can re-identify 87% of individuals through the use of three quasi-identifiers: zip code, birth date, and gender. A real-world consequence of inadequate governance structures can be seen in Google's Project Nightingale (2019), which collected 50 million patient records without obtaining explicit consent from the persons involved.

4.3 Recommendations

Mandatory algorithmic impact assessments, which are analogous to clinical drug trials, on all AI tools prior to entering clinical use, including disaggregating results from the evaluations based on demographics and conducting a third-party independent audit on the tool's performance.

5 Technical Implementation

5.1 Code quality approach

Again all five python libraries were developed under uniform professional guidelines. Every file starts with a module-level docstring that explains what the file does and its inputs, outputs, and rubric alignment. Functions use single responsibility, are type annotated using native hints from Python 3.14.0 (list[str], dict[str, float], -> None), and named sufficiently to provide self-documenting code. We share the helper safe_action() in main. py and request_with_retry() in the scraper provide centralised error handling, instead of copying try except blocks across the entire codebase. Every output is stored in nicely structured subdirectories (data/, visualizations/) where the path is explicitly managed with os. makedirs(exist_ok=True).

6 Reflection

The coursework solidified practical competency across the entire data science lifecycle — from object-oriented system design to ethical AI analysis. The implementation of the UMS led to a better awareness of property decorators and encapsulation in Python 3.14.0, while the data pipeline reinforced the knowledge that most work done in reality is cleaning not modelling. Derivatives, Random Forest regression + scraping abstractions will follow. These skills correlate well with healthcare systems, e-commerce platforms, and AI governance frameworks in industry practice.

7 Appendices

https://github.com/lahiruweerasingha/coursework_COMScDS252P_014/