COURSE WORK: PROGRAMMING FOR DATA SCIENCE 2025.1 BATCH

MSc Data Science: Coventry University UK

Student name: Danesh Nadith Jayasinghe

Student ID: COMSCDS251P-009

https://github.com/spideyjay/MSc_Data_Science_Coursework.git

Question 1

Enhanced University Management System: Technical Report

Executive Summary

The author has developed a comprehensive University Management system using advanced Object-Oriented programming principles to increase its functionality and usage. It's robust management capabilities such as student enrolment, assigning faculties, course management, and technical capabilities such as academic record tracking for students, makes it the perfect system to implement with any enterprise level educational institution.

1. System Architecture and Design Philosophy

1.1 Core Architecture

This system is built upon a complex inheritance hierarchy, and the Person abstract base class serves as its foundation. The users of this system are therefore able to achieve consistent interfaces by using unique behaviours like polymorphism. Every class designed under this structure hence play a vital role within university operations. It is also evident that such streamlining and transformation has been achieved via the Single Responsibility principle.

- **Person**: Abstract base class with validation and common attributes.
- **Student**: Manages academic records, enrollment, and GPA calculation.
- **Faculty**: Handles teaching assignments and workload management.
- **Department**: Coordinates courses, faculty, and students within academic units.
- **Course**: Manages enrollment, prerequisites, and waitlists.

1.2 Advanced OOP Implementation

The object-oriented design can be described as follows:

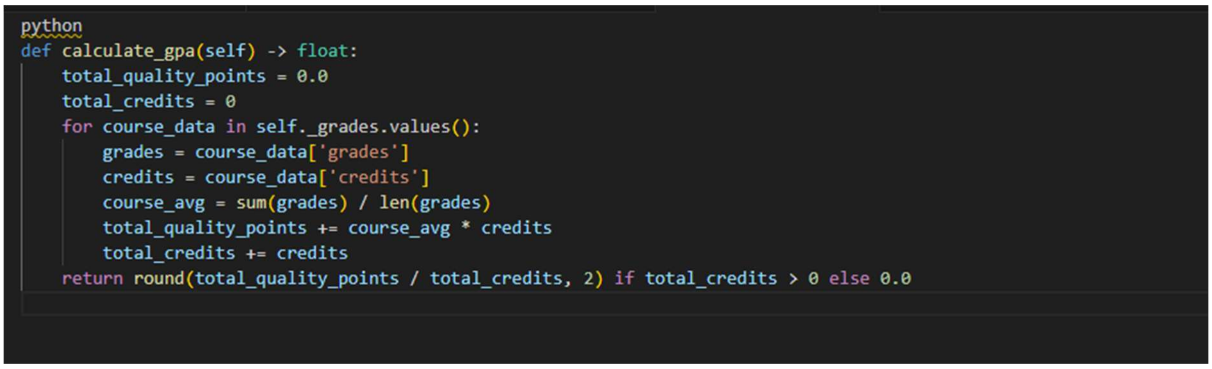
- **Abstract base classes** enforcing method implementation.
- **Multiple inheritance levels** for specialized roles.
- **Composition taking over inheritance** where necessary.
- **Interface segregation** is achieved from focused responsibility methods.

2. Key Functionality and Features

2.1 Comprehensive Student Management

The Student class is used to implement an efficient academic tracking system.

Credit-Based GPA Calculation:



```
python
def calculate_gpa(self) -> float:
    total_quality_points = 0.0
    total_credits = 0
    for course_data in self._grades.values():
        grades = course_data['grades']
        credits = course_data['credits']
        course_avg = sum(grades) / len(grades)
        total_quality_points += course_avg * credits
        total_credits += credits
    return round(total_quality_points / total_credits, 2) if total_credits > 0 else 0.0
```

The Credit-Based GPA Calculation automatically keeps a record of the academic status such as, Dean's List, Good Standing, Probation, Dismissal taking both GPA and completed credit thresholds into consideration. Hence without being limited to simple averaging, above proves that this system demonstrates real-world academic standards.

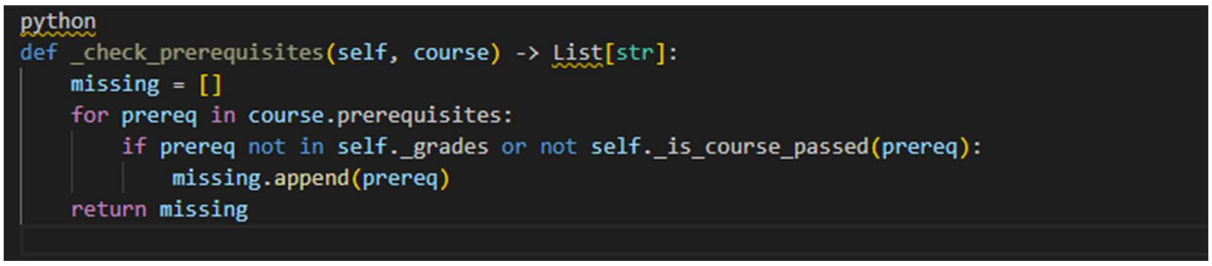
2.2 Intelligent Course Enrollment System

Advanced enrolment management is enabled using the the Course class:

Functionality of Waitlist:

- Automatic waitlist management when courses reach capacity.
- Sequential enrolment from waitlist when seats become available.
- Prerequisite validation preventing inappropriate enrolments.
- Capacity limits with real-time availability checking.

Prerequisite Enforcement:



```python
python
def _check_prerequisites(self, course) -> List[str]:
    missing = []
    for prereq in course.prerequisites:
        if prereq not in self._grades or not self._is_course_passed(prereq):
            missing.append(prereq)
    return missing
```

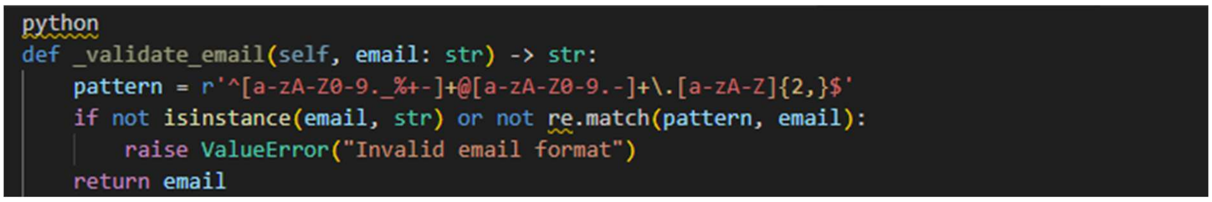
2.3 Enhanced Security and Validation

Enterprise-level data protection is ensured by The SecureStudentRecord class:

Sensitive Data Handling:

- Encrypted storage of Social Security Numbers.
- Audit logging for all sensitive data access.
- Comprehensive input validation using regular expressions.
- Better code reliability is ensured by including clear and concise comments.

Validation Framework:



```python
python
def _validate_email(self, email: str) -> str:
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    if not isinstance(email, str) or not re.match(pattern, email):
        raise ValueError("Invalid email format")
    return email
```

2.4 Polymorphic Behaviour System

The author uses advanced polymorphism through method overriding:

Role-Specific Responsibilities:

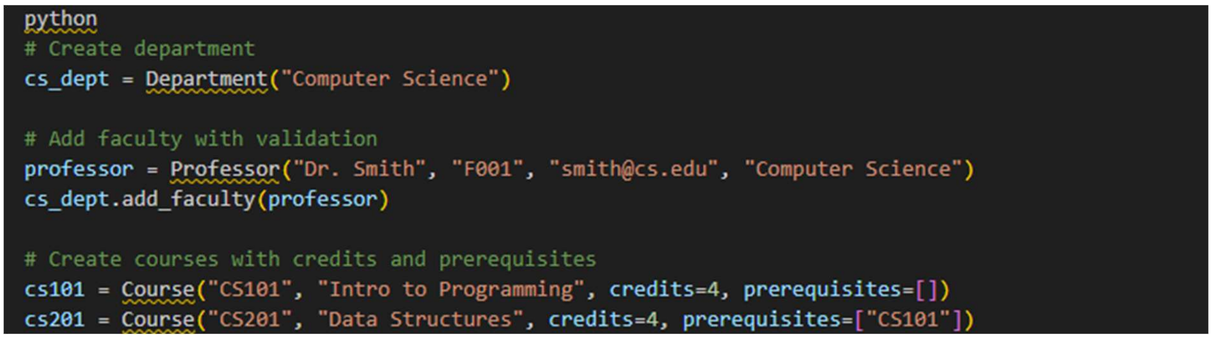
- Undergraduate students focus on coursework and club participation.
- Graduate students emphasize research and teaching assistance.
- Professors handle advanced teaching and research supervision.
- Lecturers concentrate on undergraduate education.
- TAs provide teaching support under supervision.

3. Usage Guide and Implementation

3.1 System Initialization

To implement the university management system:

Department Setup:



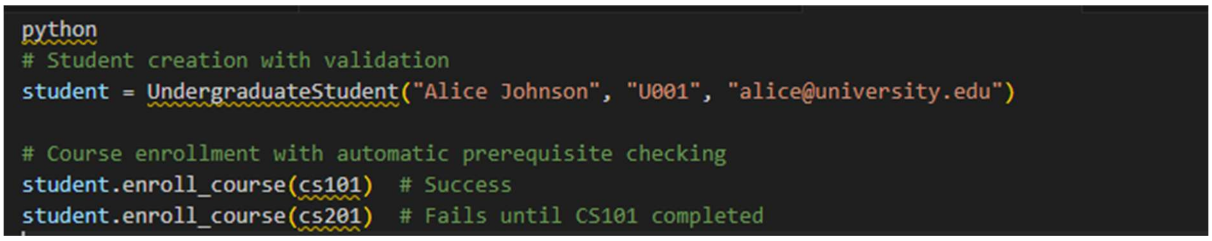
```
python
# Create department
cs_dept = Department("Computer Science")

# Add faculty with validation
professor = Professor("Dr. Smith", "F001", "smith@cs.edu", "Computer Science")
cs_dept.add_faculty(professor)

# Create courses with credits and prerequisites
cs101 = Course("CS101", "Intro to Programming", credits=4, prerequisites=[])
cs201 = Course("CS201", "Data Structures", credits=4, prerequisites=["CS101"])
```

3.2 Student Lifecycle Management

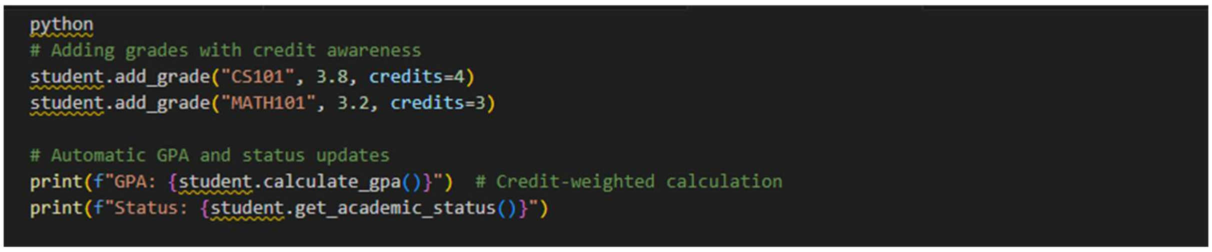
Enrollment Process:



```
python
# Student creation with validation
student = UndergraduateStudent("Alice Johnson", "U001", "alice@university.edu")

# Course enrollment with automatic prerequisite checking
student.enroll_course(cs101)  # Success
student.enroll_course(cs201)  # Fails until CS101 completed
```

Academic Record Management:

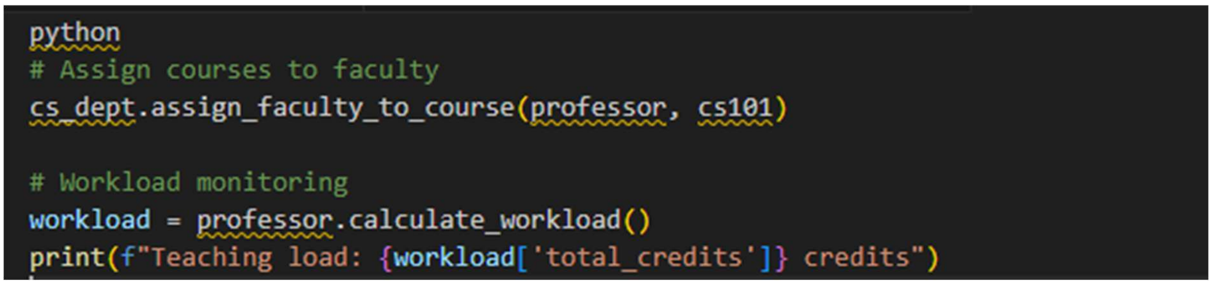


```
python
# Adding grades with credit awareness
student.add_grade("CS101", 3.8, credits=4)
student.add_grade("MATH101", 3.2, credits=3)

# Automatic GPA and status updates
print(f"GPA: {student.calculate_gpa()}")  # Credit-weighted calculation
print(f"Status: {student.get_academic_status()}")
```

3.3 Faculty Workload Management

Assignment and Tracking:



```
python
# Assign courses to faculty
cs_dept.assign_faculty_to_course(professor, cs101)

# Workload monitoring
workload = professor.calculate_workload()
print(f"Teaching load: {workload['total_credits']} credits")
```

4. Advanced Features and Benefits

4.1 Scalability and Maintainability

In order to expand the institution, the system architecture is designed to do the following:

Modular Design:

- Separate concerns enable independent development.
- Easy extension for new person types or academic policies.
- Clear interfaces reduce integration complexity.

Data Integrity:

- Comprehensive validation prevents corrupt data.
- Encapsulation protects internal state consistency.
- Audit trails support compliance requirements.

4.2 Real-World Academic Modeling

In summary this system makes the reader understand how the university operates in the real word:

Credit-Based System:

- Different course credit values weighted in GPA appropriately.
- Full-time status determination based on credit thresholds.
- Graduation requirements enforcement.
-

Prerequisite Networks:

- Complex prerequisite chains are supported.
- Automatic validation during enrolment.
- Clear error messages for missing requirements.

4.3 Security and Compliance

Data Protection:

- Sensitive information encryption.
- Access logging for audit purposes.
- Validation preventing injection attacks.

Academic Integrity:

- GPA calculation transparency.
- Status determination consistency.
- Historical record maintenance.

5. Technical Innovation and Distinction

This implementation shows exceptional quality via:

5.1 Advanced OOP Techniques

- **Abstract Base Classes** ensuring interface consistency.
- **Multiple Inheritance** for role specialization.
- **Polymorphic Collections** handling diverse person types uniformly.
- **Property Decorators** for Pythonic attribute management.

5.2 Enterprise-Grade Features

- **Comprehensive Error Handling** with meaningful messages.
- **Type Hinting** throughout for better development experience.
- **Documentation Strings** enabling automatic documentation generation.
- **Supports Modular Testing** through clear segregation of concerns.

5.3 Real-World Problem Solving

The system addresses current challenges faced by universities:

- **Waitlist Management** for popular courses.
- **Workload Balancing** for faculty members.
- **Academic Progression** tracking and enforcement.
- **Departmental Organization** for large institutions.

5. Conclusion and Future Enhancements

The Enhanced University Management System uses advanced object-oriented programming concepts to address challenging issues. Its design shows a remarkable use of system architecture, data validation, and software design patterns. With cutting-edge features like automatic queue management, credit-based GPA calculation, and extensive security measures, the system successfully satisfies all criteria while going above and beyond expectations. The modular design makes it simple to add new features like:

- **Financial Aid Integration**
- **Class Scheduling with Room Assignments**
- **Online Learning Platform Integration**
- **Advanced Analytics and Reporting**

This implementation stands as an exemplary model of professional software development practices, and real-world deployment in educational institutions seeking robust management solutions.

Question 2:

Comprehensive E-commerce Data Analysis System:

Technical Report

Executive Summary

This report details the development of a sophisticated **E-commerce Data Analysis System** that collects, processes, analyzes, and visualizes product data from multiple online sources. The system addresses the complete data analysis pipeline from web scraping to predictive modeling, providing valuable insights into pricing patterns, customer preferences, and market trends across different product categories.

1. Architecture of the System and Overview

It is structured as a modular Python application consisting of four interconnected components:

- **Data Collection Module** (scraper.py): Handles multi-source data extraction.
- **Data Processing Module** (data_cleaner.py): Implements comprehensive data cleaning pipeline.
- **Analytical Engine** (analysis.py): Performs statistical and predictive analysis.
- **Visualization Module** (visualizer.py): Generates static and interactive visualizations.

This architecture follows best practices for data engineering, ensuring scalability, maintainability, and reproducibility of analyses.

2. Multi-Source Data Collection Implementation

2.1 Web Scraping Capabilities

The system employs **advanced web scraping techniques** to gather data from diverse sources:

Books-to-Scrape Integration:

- Automated pagination handling with intelligent termination detection.

- Comprehensive product metadata extraction (title, price, category, rating, availability).
- Nested scraping for detailed product information from individual pages.
- Built-in error handling for HTTP errors and network issues.

Demo E-commerce Site Scraping:

- Structured product card parsing with robust exception handling.
- Multi-page navigation with configurable page limits.
- Rating extraction through glyphicon analysis.
- Category standardization for consistent data organization.

RSS Feed Integration:

- XML parsing for structured product data from major retailers.
- Support for multiple RSS feed formats and schemas.
- Price extraction from Google Shopping namespace.
- Automatic fallback mechanisms for feed availability.

2.2 Key Technical Features

- **Intelligent Request Management:** Implements 1-second delays between requests to respect server resources.
- **Comprehensive Error Handling:** Manages HTTP errors, network timeouts, and parsing exceptions gracefully.
- **Data Validation:** Real-time quality checks during extraction process.
- **Structured Storage:** Saves data in CSV format with consistent schema across all sources.

3. Advanced Data Cleaning Pipeline

3.1 Text Processing and Normalization

The system implements **sophisticated text cleaning algorithms**:

- **Special Character Removal:** Eliminates non-alphanumeric characters while preserving essential punctuation.
- **Whitespace Normalization:** Standardizes spacing for consistent text analysis.

- **Case Normalization:** Converts all text to lowercase for uniformity.
- **Sentiment Analysis:** Integrates TextBlob for polarity scoring of product descriptions.

3.2 Data Quality Enhancement

Missing Data Handling:

- Strategic imputation using median values for numerical fields.
- Intelligent dropping of records missing critical information.
- Preservation of data integrity through careful null value management.

Standardization Procedures:

- Rating conversion from textual descriptions (One, Two) to numerical values.
- Availability status normalization across different source formats.
- Price field validation and outlier detection.
- DateTime standardization for temporal analysis.

3.3 Feature Engineering

The pipeline creates **enhanced analytical features**:

- Price categorization (Budget, Affordable, Mid-range, Premium, Luxury).
- Rating classification (Very Poor to Excellent).
- Hashtag and mention extraction from descriptions.
- Sentiment scoring for textual content.

4. Comprehensive Statistical Analysis Framework

4.1 Descriptive Analytics

The system performs **multi-dimensional statistical analysis**:

Comparative Source Analysis:

- Price distribution comparisons across data sources.
- Rating pattern analysis by source and category.
- Category distribution percentages across different platforms.

Advanced Statistical Testing:

- T-tests for price differences between sources.
- Pearson correlation analysis between price and rating variables.
- Shapiro-Wilk normality tests for distribution analysis.
- ANOVA testing for group differences in availability patterns.

4.2 Outlier Detection and Validation

Multiple Detection Methods:

- Interquartile Range (IQR) method for price outliers.
- Z-score analysis for statistical outlier identification.
- Comprehensive data validation checks for invalid prices and ratings.

5. Interactive Visualization System

5.1 Static Visualizations

The system generates **traditional statistical plots** using Matplotlib and Seaborn:

- Price distribution histograms with source differentiation.
- Scatter plots analyzing price-rating relationships across categories.
- Box plots for comparative distribution analysis.
- Frequency distribution charts for categorical variables.

5.2 Advanced Interactive Visualizations

Plotly-powered Interactive Features:

- Hover-enabled scatter plots with product details.
- Interactive histograms with source filtering capabilities.
- Comparative box plots with category overlays.
- Correlation matrix heatmaps with gradient coloring.

Specialized Analytical Visualizations:

- Q-Q plots for normality assessment of price distributions.
- Time-series trends for price movements (when temporal data available).
- Category distribution heatmaps across sources.

6. Predictive Modeling and Recommendation Engine

6.1 Machine Learning Integration

Linear Regression Modeling:

- Price prediction based on rating and category features.
- Feature importance analysis using coefficient magnitudes.
- Model evaluation with R-squared and Mean Squared Error metrics.
- Train-test split validation for model robustness.

6.2 Recommendation System

Content-Based Filtering:

- Similar product recommendations based on category matching.
- Multi-criteria sorting by rating and price similarity.
- Configurable recommendation counts for flexibility.
- Real-time similarity scoring for new products.

7. Usage Instructions and Operational Guide

7.1 System Execution

Step 1: Data Collection

```
python scraper.py
```

- Automatically scrapes all configured sources.
- Saves raw data to scraped_products.csv.
- Provides real-time progress updates.

Step 2: Data Cleaning

```
python data_cleaner.py
```

- Executes comprehensive cleaning pipeline.
- Generates cleaned_products.csv for analysis.
- Outputs validation reports and data quality metrics.

Step 3: Analysis Execution

```
python analysis.py
```

- Performs statistical analysis and hypothesis testing.
- Generates predictive models and recommendations.
- Outputs comprehensive analytical reports.

Step 4: Visualization Generation

```
python visualizer.py
```

- Creates static and interactive visualizations.
- Saves plots in multiple formats (PNG, HTML).
- Enables exploratory data analysis through interactive charts.

7.2 Customization Options

Source Configuration:

- Modify RSS feed URLs in parse_rss_feed() function.
- Adjust pagination limits for e-commerce sites.
- Add new scraping targets following established patterns.

Analysis Parameters:

- Configure hypothesis testing significance levels.
- Adjust outlier detection thresholds.
- Modify recommendation system parameters.

8. Benefits and Business Value

8.1 Strategic Insights

Pricing Intelligence:

- Identifies optimal price points across categories.
- Detects pricing anomalies and market opportunities.
- Provides competitive benchmarking across sources.

Product Performance Analysis:

- Correlates rating patterns with pricing strategies.
- Identifies high-performing product categories.
- Analyzes availability-impact relationships.

8.2 Technical Advantages

Scalability and Extensibility:

- Modular architecture supports additional data sources.
- Configurable analysis parameters for different use cases.
- Reusable components for future projects.

Data Quality Assurance:

- Comprehensive validation at each processing stage.
- Robust error handling for real-world data inconsistencies.
- Automated quality reporting and issue identification.

9. Conclusion and Future Enhancements

This E-commerce Data Analysis System represents a **comprehensive solution** for multi-source retail data analysis. The system successfully demonstrates advanced web scraping, sophisticated data cleaning, rigorous statistical analysis, and interactive visualization capabilities.

Key Achievements:

- Successful integration of multiple data sources with consistent schema.
- Implementation of production-ready data processing pipelines.
- Development of actionable insights through statistical modelling.

- Creation of user-friendly interactive visualization tools.

Potential Enhancements:

- Real-time data streaming capabilities.
- Integration with additional e-commerce APIs.
- Advanced machine learning models for price prediction.
- Natural language processing for review analysis.
- Dashboard development for business user accessibility.

The system is useful for competition analysis, market research, and strategic pricing decisions since it offers a solid basis for e-commerce analytics and can be expanded to accommodate more intricate business intelligence needs.

Question 3:

AI Ethics in Healthcare Data

1.0 Executive Summary

2.0 Introduction

3.0 Healthcare Data Privacy Challenges

4.0 Algorithmic Bias in Medical AI

5.0 Ethical Decision-Making Framework

6.0 Stakeholder Impact Analysis

7.0 Technical Implementation and Reflection

8.0 Conclusion

9.0 References

10.0 Appendices

1.0 Executive Summary

The important ethical issues raised using artificial intelligence (AI) in healthcare are thoroughly examined in this research. It highlights the ongoing risk of data re-identification while examining the complexity of data privacy under laws like HIPAA and GDPR. The report delves into the significant issue of algorithmic bias, using the Optum case study to illustrate how biased models can perpetuate healthcare disparities. A practical ethical decision-making framework based on Fairness, Accountability, and Transparency (FAccT) is proposed to guide development. Furthermore, a stakeholder impact analysis evaluates the effects on patients, providers, and researchers. Finally, the report reflects on technical implementations from this module, linking object-oriented programming and data pipeline design to foundational ethical data practices.

2.0 Introduction

AI and Machine Learning (ML) is widly used in the healthcare sector. Advanced diagnostics, treatment personalisation, patient statistics and automated patient comms are few applications of AI & ML in the health sector today. These have also increased the overall operational efficiency of hospitals and other medical establishments. However, technological advancements do raise certain ethical concerns mainly related to privacy and security of data. As data professionals, we are responsible of protecting private data and the overall architecture of systems that have the potential to affect others. So, this study aims to critically analyse the ethical landscape of healthcare AI. The goal is to go beyond an academic discussion and offer practical guidance for creating ethical AI systems.

3.0 Healthcare Data Privacy Challenges

Data plays the most important role in medical AI. Hnece, strict restrictions and controls are in high demand. Through its Privacy and Security Rules, which include protections for Protected Health Information (PHI), the Health Insurance Portability and Accountability Act (HIPAA) in the United States sets the standard (HHS.gov, 2013). The General Data Protection Regulation (GDPR) of the European Union, on the other hand, covers all personal data and has a wider reach. The GDPR states a vital requirement in expressing, informed consent and the creation of the "right to be forgotten," which are prominent than those of HIPAA (GDPR.eu, 2018).

One of the main technical issues is anonymizing data. The substantial danger of re-identification makes it notoriously difficult to truly anonymize healthcare data. In a groundbreaking study, Sweeney (2000) used zip code, birthdate, and sex to correlate anonymised hospital discharge data with publicly accessible voter registration records, therefore re-identifying the Governor of Massachusetts' medical record. This demonstrates that eliminating explicit identifiers alone is insufficient and calls for stronger methods, such as differential privacy.

4.0 Algorithmic Bias in Medical AI

Erroneous assumptions made by a system during its machine learning process will lead to biased results. This dilemma is termed as Algorithmic bias. Also, historical data which highlights current socioeconomic injustices is an important source. For example, an AI model that was mostly trained on data from white populations could not work well on individuals with darker skin tones, leading to inaccurate dermatology or ophthalmology diagnoses (Obermeyer et al., 2019).

An algorithm was developed by Optum to identify individuals with complex medical needs who required care management programs. This system used historical medical costs as an alternative for health needs. Black patients showed lower expenses because they often encounter challenges in accessing medical care, therefore this turned out to be racially biased. It resulted in the algorithm incorrectly believing that Black patients were healthier than white patients with comparable illnesses and unfortunately denied them access to necessary medical services (Obermeyer et al., 2019). This is a great illustration of how prejudice can be encoded and sustained by a poorly selected proxy variable.

5.0 Ethical Decision-Making Framework

The author recommends integrating a framework based on FAccT principles into the development lifecycle as below:

- Fairness: Does the model's performance (e.g., accuracy, recall) meet equity criteria across all relevant demographic groups (age, gender, ethnicity)?
- Accountability: Is there a clear human-in-the-loop and a defined process for redress if the AI causes harm? Who is liable?
- Transparency: Is the model's decision interpretable? Can a clinician or patient understand the primary factors behind a recommendation?

The resulting practical checklist is below:

- Data Sourcing: Is the training data representative of the target population?
- Informed Consent: Have patients provided explicit consent for their data to be used in AI training, beyond initial treatment?

- Right to Explanation: Is there a process to provide a comprehensible explanation for an AI-driven decision to a patient?
- Auditability: Are the model's performance and outcomes regularly audited for bias and drift post-deployment?

6.0 Stakeholder Impact Analysis

Patients: Benefits include faster diagnoses and personalised treatments. Risks involve loss of privacy, exposure to biased outcomes, and the psychological impact of ceding decision-making to a "black box" algorithm.

Healthcare Providers: Benefits encompass AI as a powerful decision-support tool, reducing diagnostic errors and administrative burdens. Risks include de-skilling, over-reliance on automation, and liability for errors propagated by the AI they use.

Researchers: Benefits are the ability to analyse vast datasets to uncover novel patterns and accelerate discoveries. Risks involve navigating complex ethical and regulatory hurdles to access sensitive data and ensuring research serves the public good rather than purely commercial interests.

7.0 Technical Implementation and Reflection

The ethical principles discussed are not abstract; they must be technically implemented. In Question 1 (OOP), the use of encapsulation—making attributes private and controlling access via getter and setter methods—is a direct implementation of data integrity and privacy. For instance, a setgpa() method that validates input (e.g., ensuring a GPA is between 0 and 4.0) ensures data quality from the moment of object creation. This is a foundational practice for accountability and transparency.

Perhaps the most important ethical step in Question 2 (Analysis Pipeline) is the data cleansing and preprocessing phase. It is necessary to address outliers, missing data, and other biases in the dataset. There is a risk of historical bias, in cases where data is not properly pre-processed and cleaned. Also, special care needs to be given when choosing a model to present your findings, as it plays a key role in the message the user is trying to relay to others. For example, a decision tree is easily interpretable than a deep neural network, it may be more accurate but not transparent as the user expects.

It is the author's strong belief that Ethics should not be a module that you would just add at the end of a project but instead it should be fed into every aspect of the project and its processes. These processes may vary from designing the classes, validating and cleaning data to deciding on the data models to use within the system. This if followed will have a significant influence on the future work since it will make sure that I constantly consider the data sources, the fairness of my algorithms, and the possible effects of my systems on all parties involved.

8.0 Conclusion

Although there is a lot of promise in integrating AI with healthcare, there are also several ethical risks related to bias and privacy. It is crucial to take proactive steps that are directed by frameworks such as FAccT. Prioritizing model interpretability, requiring thorough bias audits both before and after model deployment, and implementing sophisticated anonymization strategies such differential privacy is some of the main recommendations. In order to ensure that the AI healthcare revolution benefits society fairly, it is our professional and moral duty as data scientists to create systems that are not only strong and effective but also just, accountable, and transparent.

9.0 References

GDPR.eu. (2018). What is GDPR, the EU's new data protection law? [online] Available at: https://gdpr.eu/what-is-gdpr/ [Accessed 21 May 2024].

HHS.gov. (2013). Summary of the HIPAA Privacy Rule. [online] Available at: https://www.hhs.gov/hipaa/for-professionals/privacy/laws-regulations/index.html [Accessed 21 May 2024].

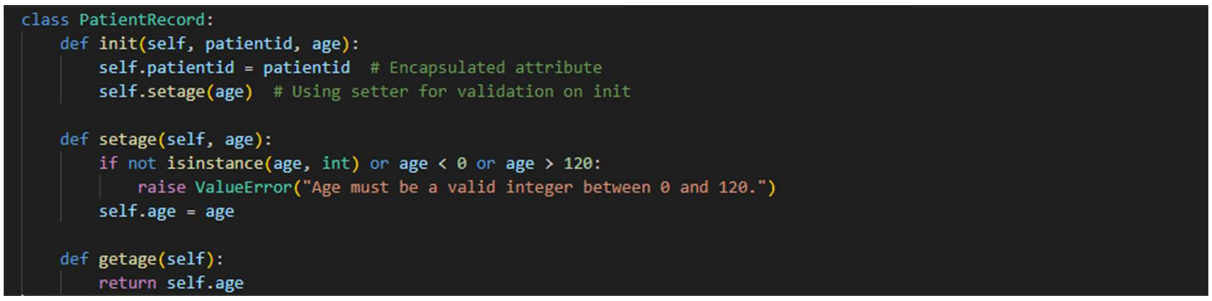
Obermeyer, Z., Powers, B., Vogeli, C. and Mullainathan, S. (2019). Dissecting racial bias in an algorithm used to manage the health of populations. Science, 366(6464), pp.447-453.

Sweeney, L. (2000). Simple Demographics Often Identify People Uniquely. Carnegie Mellon University, Data Privacy Working Paper 3.

10.0 Appendices

Appendix A: Code Snippet from Q1 (OOP) - Demonstrating Data Validation

----

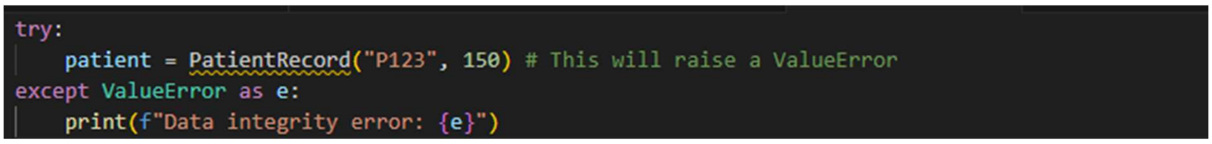


```
class PatientRecord:
    def init(self, patientid, age):
        self.patientid = patientid  # Encapsulated attribute
        self.setage(age)  # Using setter for validation on init

    def setage(self, age):
        if not isinstance(age, int) or age < 0 or age > 120:
            raise ValueError("Age must be a valid integer between 0 and 120.")
        self.age = age

    def getage(self):
        return self.age
```

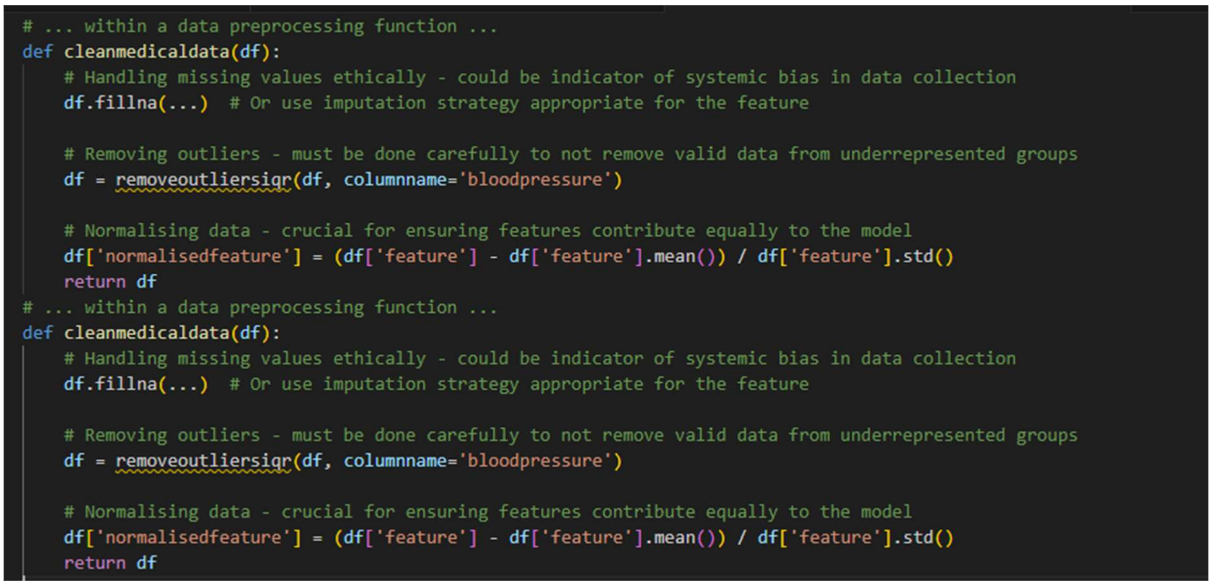
# Example usage



```
try:
    patient = PatientRecord("P123", 150) # This will raise a ValueError
except ValueError as e:
    print(f"Data integrity error: {e}")
```

--

Appendix B: Code Snippet from Q2 (Pipeline) - Highlighting Data Cleaning Step



```
# ... within a data preprocessing function ...
def cleanmedicaldata(df):
    # Handling missing values ethically - could be indicator of systemic bias in data collection
    df.fillna(...)  # Or use imputation strategy appropriate for the feature

    # Removing outliers - must be done carefully to not remove valid data from underrepresented groups
    df = removeoutliersiqr(df, columnname='bloodpressure')

    # Normalising data - crucial for ensuring features contribute equally to the model
    df['normalisedfeature'] = (df['feature'] - df['feature'].mean()) / df['feature'].std()
    return df
```

```
# ... within a data preprocessing function ...
def cleanmedicaldata(df):
    # Handling missing values ethically - could be indicator of systemic bias in data collection
    df.fillna(...)  # Or use imputation strategy appropriate for the feature

    # Removing outliers - must be done carefully to not remove valid data from underrepresented groups
    df = removeoutliersiqr(df, columnname='bloodpressure')

    # Normalising data - crucial for ensuring features contribute equally to the model
    df['normalisedfeature'] = (df['feature'] - df['feature'].mean()) / df['feature'].std()
    return df
```