TECHNICAL REPORT

Coursework_COMScDS252P-010 – Amalka Chandrasiri

Contents

Executive Summary

This coursework covers 3 main areas:

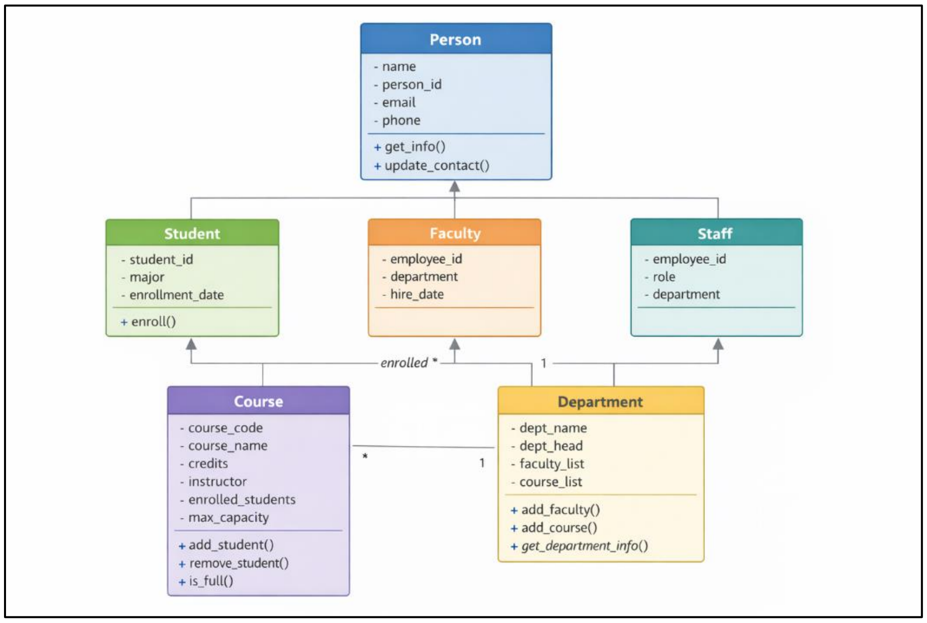
1. University Management System
2. E-commerce Data Analysis
3. AI Ethics in Healthcare

This report covers the fundamentals of Object-Oriented Programming [OOP] concepts, web scraping, data analysis, visualization, and prediction. It also introduces AI ethics within the context of healthcare, discussing the impact of AI models, data privacy concerns, and established protocols designed to protect sensitive data.

Question 1: OOP Implementation

Architecture And Class Hierarchy

The goal of implementing the university system was to support different roles, responsibilities, and facilities by maintaining structured information based on each role. This was achieved using a hierarchical design. At the top level, the basic details common to all roles were organized under the Person parent in the hierarchy. Each specific role was then implemented as a child class at the next level, inheriting the shared attributes while defining its own responsibilities. In addition, two separate classes: Department and Course. These were created to effectively manage and represent faculty-related information.



Design Decisions

All basic information validations common to every role were implemented at the Person level to avoid duplication. Each class was defined in a separate .py file to improve code maintainability, allowing modules to be imported as needed. The entire solution was organized within a single folder to further enhance structure and simplify maintenance.

Key OOP Features Demonstrated

Inheritance

In the hierarchical structure, the parent class stores all the basic information that is common across roles. The child classes inherit these attributes and behaviors using the super () method, while also defining their own specific functionalities. This approach improves code reusability and maintains a clear relationship between classes within the hierarchy.

Polymorphism

get_info() is a common method implemented across all classes in the hierarchy. However, the output of this method varies depending on the class. Each object displays its own relevant information, demonstrating polymorphism through overriding methods.

Encapsulation

All attributes (data) and behaviors (methods) were encapsulated within their respective classes. Sensitive information, such as student ID and student GPA, was declared as private using access control mechanisms. These variables cannot be accessed directly from outside the class, ensuring data protection and controlled modification through designated methods. This enhances security and maintains the integrity of the system.

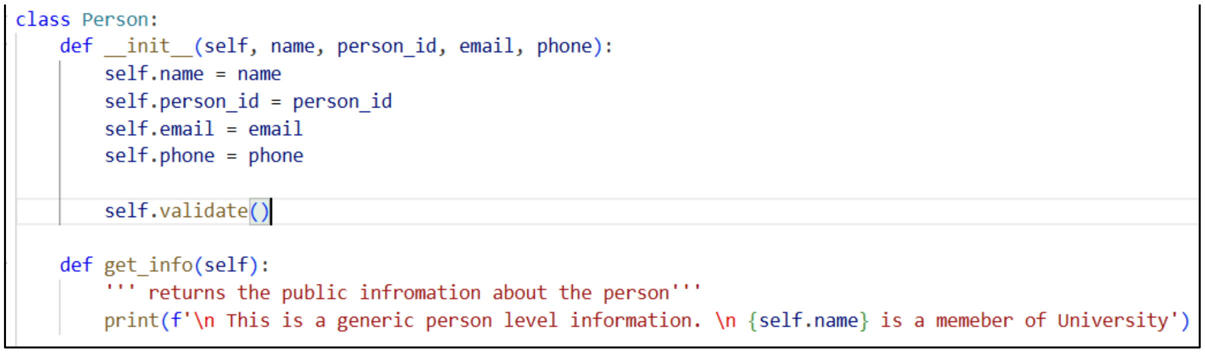
Abstraction

Several operations, such as enrolling a student, adding a course, and calculating GPA, were implemented through dedicated methods. The internal logic of these operations is hidden from the end user. Users can perform tasks simply by calling the relevant method without needing to understand how the process is executed internally.

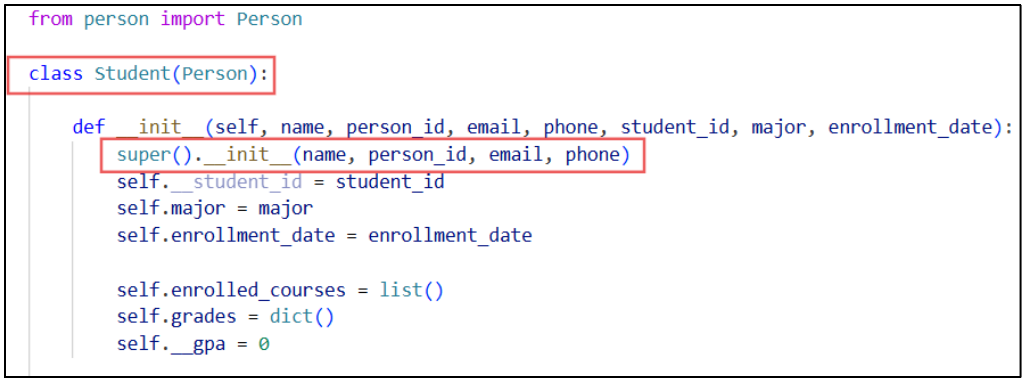
Selected Code Snippets with Explanations

Inheritance

The Student class inherits from the Person class (as indicated in the class definition). Therefore, the student obtains all attributes and methods of Person, such as name, person_id, email, phone, and the get_info() method. In addition, it defines its own attributes and methods, including major, grades, and calculate_gpa().



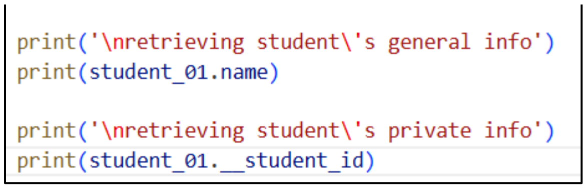
```
class Person:
    def __init__(self, name, person_id, email, phone):
        self.name = name
        self.person_id = person_id
        self.email = email
        self.phone = phone

        self.validate()

    def get_info(self):
        ''' returns the public infromation about the person'''
        print(f'\n This is a generic person level information. \n {self.name} is a memeber of University')
```



```
from person import Person

class Student(Person):

    def __init__(self, name, person_id, email, phone, student_id, major, enrollment_date):
        super().__init__(name, person_id, email, phone)
        self.__student_id = student_id
        self.major = major
        self.enrollment_date = enrollment_date

        self.enrolled_courses = list()
        self.grades = dict()
        self.__gpa = 0
```

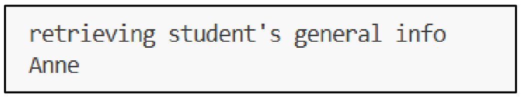
Encapsulation

Some student information, such as name, is public, while other details, such as student ID, are kept private. Making attributes private restricts direct access from outside the class, providing data protection. Attempting to access a private attribute directly results in an error, such as: AttributeError: 'Student' object has no attribute '_student_id'.

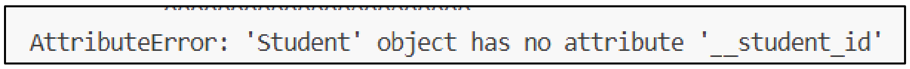


```
print('\nretrieving student\'s general info')
print(student_01.name)

print('\nretrieving student\'s private info')
print(student_01.__student_id)
```

Output



```
retrieving student's general info
Anne
```

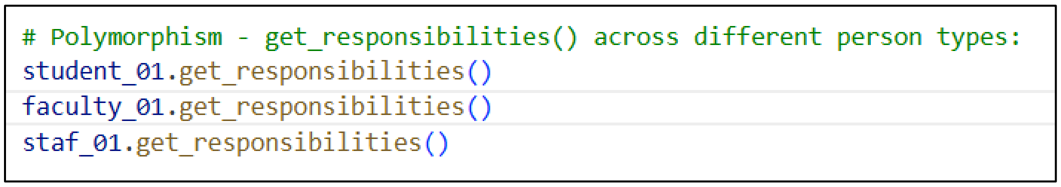


```
AttributeError: 'Student' object has no attribute '__student_id'
```

Polymorphism

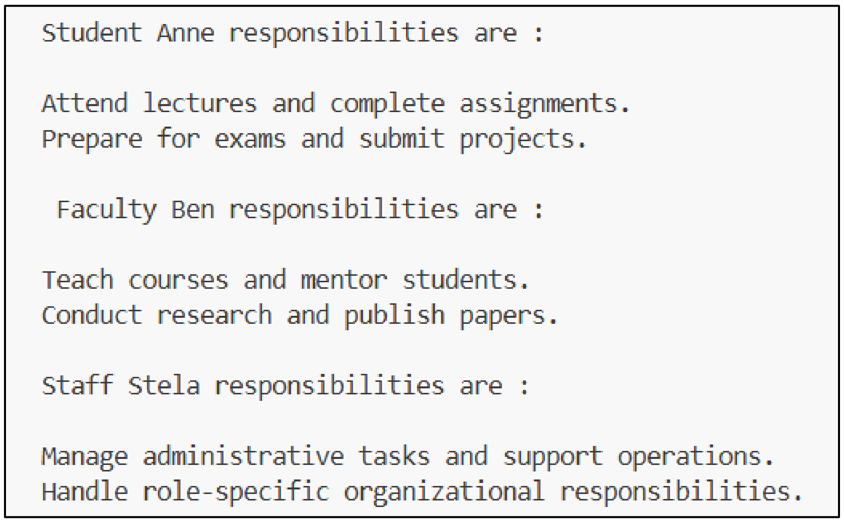
This is achieved by defining the same method name across classes, with behavior varying based on the class type.

Eg: Each class has method : get_responsibilities(), but depending on the object, the output differs.



```
# Polymorphism - get_responsibilities() across different person types:
student_01.get_responsibilities()
faculty_01.get_responsibilities()
staf_01.get_responsibilities()
```

Output



```
Student Anne responsibilities are :

Attend lectures and complete assignments.
Prepare for exams and submit projects.

 Faculty Ben responsibilities are :

Teach courses and mentor students.
Conduct research and publish papers.

Staff Stela responsibilities are :

Manage administrative tasks and support operations.
Handle role-specific organizational responsibilities.
```

Question 2: Data Analysis

Methodology And Tools

Data analysis was implemented in VS Code with a structured folder organization, assigning a separate .py file for each step: data scraping, cleaning, analysis, visualization, and prediction. Two CSV files were used—one for raw data and another for cleaned data—to enable comparison and easy access. Adding a random sleep interval between requests simulates human behavior and prevents overloading the website, protecting real users. Python libraries were used to standardize the process, including requests for downloading webpage data and BeautifulSoup, which simplifies navigation and extraction of information from HTML content efficiently.

Key Findings from Exploratory Analysis

Exploratory Data Analysis (EDA) was conducted as a step-by-step process to ensure that no information was unintentionally lost, replaced, or deleted during preprocessing. The first step was to identify the dataset size, including the number of records and attributes, before applying any modifications.

After having an idea about the size of the dataset, the next step was to clean and prepare the data for analysis. The Price (£) column, initially a string with the "£" symbol, was converted to a numeric float type to allow calculations. Duplicate and null values were assessed using df.isna(). sum(), and no missing data was found.

Two new features were created to enhance the dataset:

- Price Category: categorizes records based on price
- In Stock – Availability of the books.

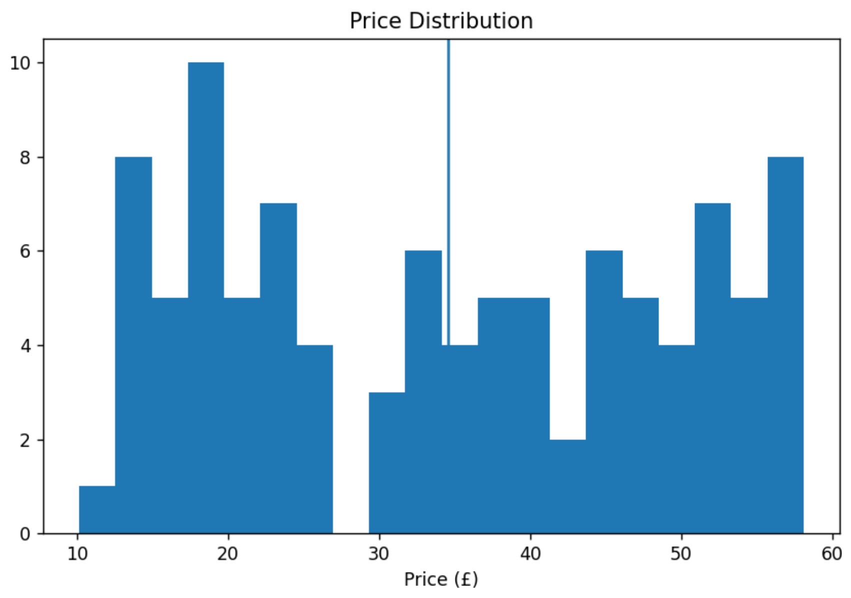
Statistical Results with Visualizations

Pandas provide built-in methods for basic statistical analysis, while SciPy. Stats library offers additional functions for statistical inference. Measures such as mean, median, mode, and standard deviation were calculated using Pandas, providing fundamental insights into the dataset.

```
print(f'\n---- Descriptive Statistics ---')
# Central tendency: mean, median, mode for prices
price_mean = round(books_data['Price (£)'].mean(), 2)
price_median = round(books_data['Price (£)'].median(), 2)
price_mode = round(books_data['Price (£)'].mode()[0], 2)
```

```
---- Descriptive Statistics ---
price mean - 34.56
price median - 34.78
price mode - 44.18
Price range is 47.95
```

This result was further validated using a histogram of the price distribution with a mean reference line. The position of the mean line aligned with the previously calculated mean value, visually confirming the statistical result.



Quantiles and interquartile range (IQR) were used to measure data spread efficiently.

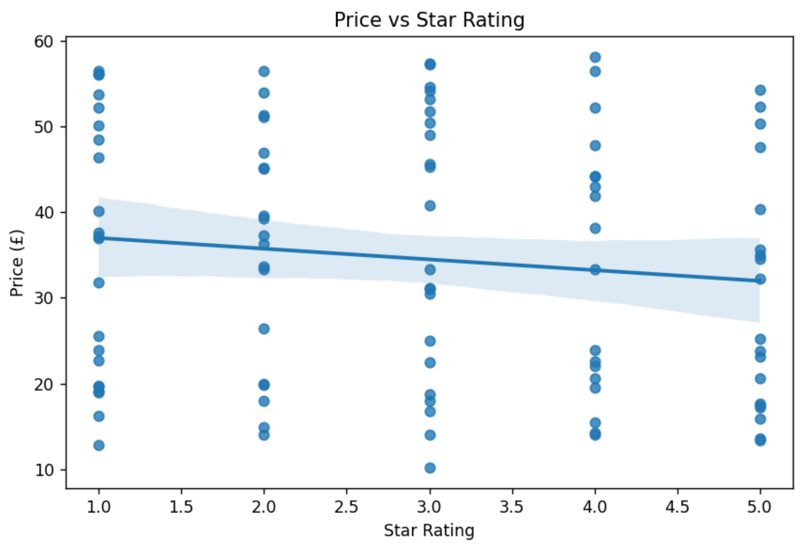
```
# Outlier Detection
Q1 = books_data['Price (£)'].quantile(0.25)
Q3 = books_data['Price (£)'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers = books_data[(books_data['Price (£)'] < lower_bound) | (books_data['Price (£)'] > upper_bound)]

print('Lower Bound for Price:', lower_bound)
print('Upper Bound for Price:', upper_bound)

if outliers.empty:
    print('There\'s no outliers for Price.')
else:
    print('Outliers are - ', outliers['Price (£)'])
```

The correlation between price and rating was calculated using pearsonr, resulting in a weak negative correlation, indicating a slight inverse relationship between the two variables.

```
# Correlation analysis: Pearson correlation between price and rating
correlation = pearsonr(books_data['Price (£)'], books_data['Star Rating'])
print('\nCorrelation between Price and Rating is :', round(correlation[0], 2))
```

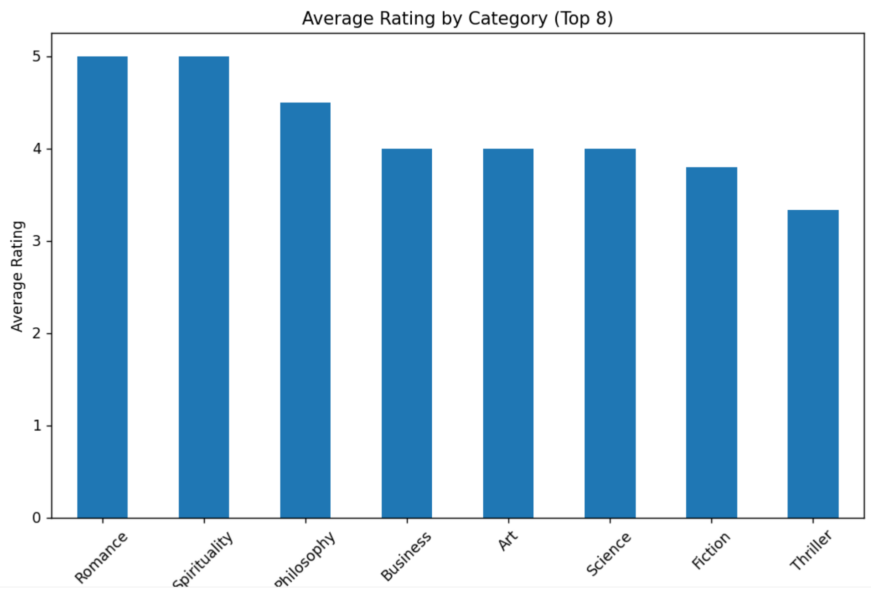
```
Correlation between Price and Rating is : -0.12
```

Which shows a weak negative correlation. This can be further confirmed with a scatter plot

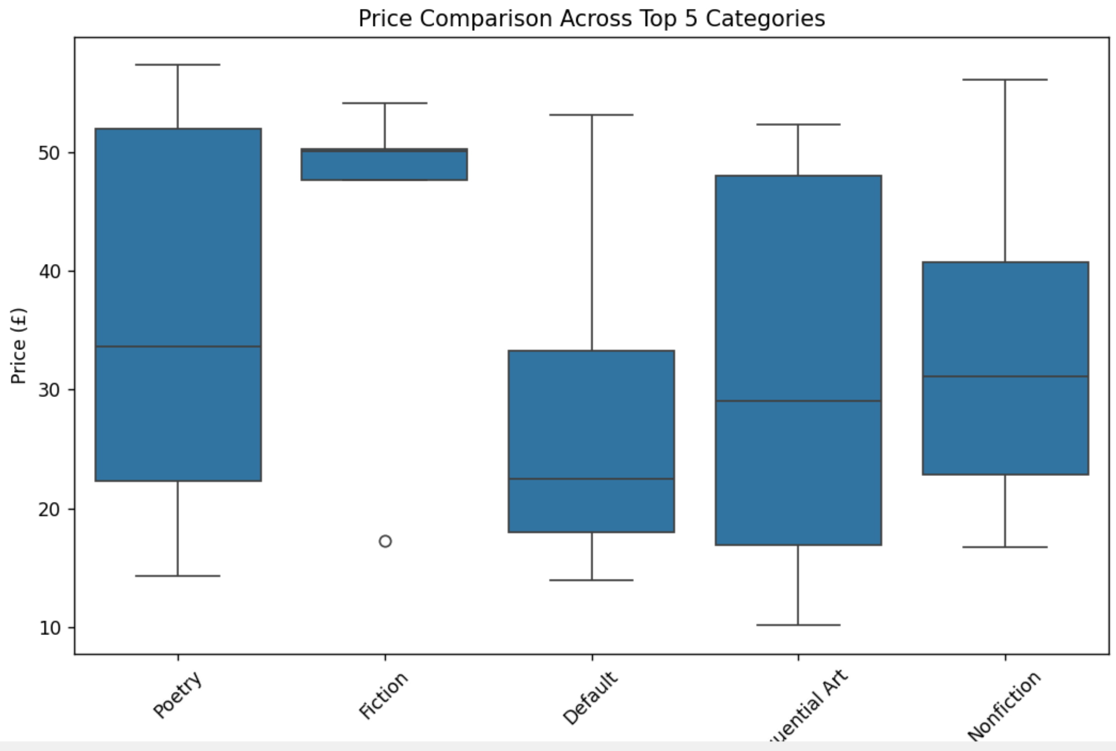


Business Insights and Interpretations

Based on the ratings by category, Romance and Spirituality show the highest average ratings, while Thriller and Fiction have comparatively lower ratings. However, overall, the ratings across all categories indicate that most books are generally well-received as they have a rating between 3 – 5.



However, when examining the prices of the top five categories based on the number of books, Romance and Spirituality are not included in the list. Based on this insight, it can be recommended that the publication of Romance and Spirituality books be increased, as these categories demonstrate higher ratings despite having fewer titles.



Question 3: Ethics Analysis

Framework summary

In healthcare data science, ethical decision-making is guided by four key principles: autonomy, non-maleficence, beneficence, and justice, which means patients should have control over their own health data and clearly understand how it is being used. With the involvement of AI, such tools must avoid causing harm, whether through privacy breaches, biased predictions, or unsafe recommendations. The goal should be to improve patient outcomes physically, mentally and overall quality of life. Fairness is also equally important at the same time, which means that AI tools should work for different demographic and socio-economic groups without any bias. These principles help ensure that healthcare AI provides a responsible and patient-focused service.

Key ethical concerns

Medical information is highly sensitive. Therefore, data privacy is very important. Bias in training data can lead to unequal treatment suggestions. If a system is not trained with a balanced dataset which reperesnt all groups of a pateints in a community, there is a risk of relying on AI outputs without further clarification or confirmation for them. Limited transparency in complex AI models can further reduce trust.

Recommendations

Healthcare AI systems should implement strong data governance policies to address these concerns. This includes encryption of sensitive data, access control, and security measures. Diverse datasets must be used to minimize bias and promote fairness. Biasness of an AL model should be continuously monitored and evaluated to detect such bias in advance. Transparent and explainable AI models should be prioritized to enhance trust and accountability. This can also include guidelines and responsibilities among healthcare providers, institutions, and developers

Technical Implementation

Code Quality Approach

The code follows PEP 8 style guidelines to ensure readability and consistency throughout the project. Docstrings were added to provide runtime support that describes the purpose of classes and methods. Comments were included where necessary to improve code usability.

Testing and validation strategies

A separate unittest class was implemented to perform basic unit testing to validate data and enhance the quality of the solution. Error handling was introduced to ensure the program continues even if issues occur during execution.

Reflection - Learning outcomes

Git with VS Code

I learned how VS Code integrates with Git, making it easy to commit changes, switch between branches, and merging updates into the main branch without needing to memorize Git commands.

Unit Testing Framework

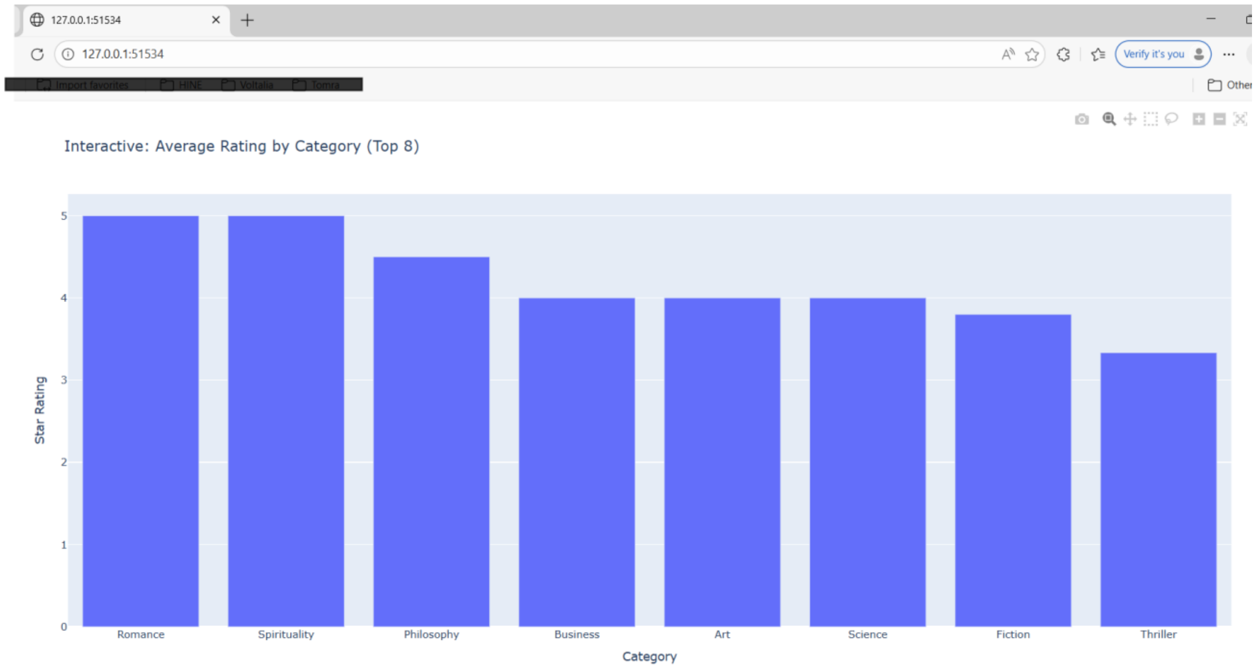
Gained hands-on experience with Python's unittest framework, which allows automated testing and ensures that methods behave as expected, making the code more robust and maintainable.

Statistical Analysis Using Python

Could gain practical experience using built-in libraries such as NumPy, Pandas, and SciPy, which provide ready-made statistical functions. It demonstrated how powerful Python is for data analysis, as existing libraries offer optimized methods for most statistical operations without needing to implement formulas manually, which makes it more time-efficient.

Visualization

Various visualizations were used to explore prices, categories, and ratings. An interactive Plotly chart with hover and zoom features provided deeper insight into advanced visualization techniques.



[redacted]