COMPREHENSIVE CA PRACTICE GUIDE

Analytics Programming & Data Visualisation (H9APDV)

Complete Practice with Original Questions + Additional Exercises + Full Solutions

---

Table of Contents

CA SET 1: RERUN ASSESSMENT (100 marks total)

Overview

This assessment covers: - **Task 1**: XML File Handling (40 marks) - Working with flights.xml - **Task 2**: Lists, Dictionaries & NumPy (30 marks) - Data structure manipulation - **Task 3**: Text Analytics with Regex (30 marks) - Review analysis

Data files provided: - `flights.xml` - Aer Lingus flight data (Dublin Rome & Amsterdam) - `reviews.txt` - Customer product reviews with ratings, hashtags, mentions

---

TASK 1: XML File Handling (40 marks)

Dataset: `flights.xml` - Contains Aer Lingus flights with: - Flight name, aircraft, departure times, routes - Origin and destination airports with IDs

Question 1.1: Safe XML Import Function (15 marks)

Write a function `load_xml(path)` that: - Opens and parses XML file safely - Handles: file not found, permission denied, malformed XML - Returns root element on success, None on failure - Prints clear error messages

What you need to know: - Use `xml.etree.ElementTree` - Try/except blocks - Different exception types (FileNotFoundError, PermissionError, ET.ParseError)

```python
import xml.etree.ElementTree as ET

def load_xml(path):
    """
    Safely load and parse an XML file.

    Args:
        path (str): Path to XML file

    Returns:
        ET.Element: Root element if successful, None if failed
    """
    try:
        # Try to parse the XML file
        tree = ET.parse(path)
        root = tree.getroot()
        print(f"  Successfully loaded XML from {path}")
        return root

    except FileNotFoundError:
        # File does not exist
        print(f"  Error: File '{path}' not found.")
        return None

    except PermissionError:
        # No read permission
        print(f"  Error: Permission denied reading '{path}'.")
        return None

    except ET.ParseError as e:
        # XML is malformed
        print(f"  Error: XML parsing failed - {e}")
        return None

    except Exception as e:
        # Catch-all for other errors
        print(f"  Unexpected error: {e}")
        return None


# TEST YOUR FUNCTION
if __name__ == "__main__":
    root = load_xml("flights.xml")
    if root is not None:
        print(f"Root tag: {root.tag}")
```

Expected output:

```
 Successfully loaded XML from flights.xml
Root tag: flights
```

Question 1.2: Extract Flight Data (15 marks)

Using the root from 1.1, extract flight information into a list of dictionaries.

Requirements: - Create list named `flights_data` - Each flight must have: date, origin_id, destination_id, aircraft_model, scheduled_departure, actual_departure - Print number of flights - Print first 3 dictionaries

Key XML structure understanding:

```
<flight>
  <date>2024-01-15</date>
  <origin id="DUB">Dublin Airport...</origin>
  <destination id="FCO">Rome Airport...</destination>
  <aircraft_model>A320-214</aircraft_model>
  <scheduled_departure>15:35</scheduled_departure>
  <actual_departure>16:07</actual_departure>
</flight>
```

Solution approach:

```
def extract_flights(root):
    """
    Extract flight data from XML root element.

    Args:
        root (ET.Element): Root element from XML

    Returns:
        list: List of flight dictionaries
    """
    flights_data = []

    # Loop over each <flight> element
    for flight in root.findall('flight'):
        # Extract basic fields
        date = flight.findtext('date', 'N/A')
        aircraft_model = flight.findtext('aircraft_model', 'N/A')
        scheduled_departure = flight.findtext('scheduled_departure', 'N/A')
```

```python
        actual_departure = flight.findtext('actual_departure', 'N/A')

        # Extract origin_id (attribute of <origin> element)
        origin_elem = flight.find('origin')
        origin_id = origin_elem.get('id', 'N/A') if origin_elem is not None else 'N/A'

        # Extract destination_id (attribute of <destination> element)
        dest_elem = flight.find('destination')
        destination_id = dest_elem.get('id', 'N/A') if dest_elem is not None else 'N/A'

        # Build dictionary
        flight_dict = {
            'date': date,
            'origin_id': origin_id,
            'destination_id': destination_id,
            'aircraft_model': aircraft_model,
            'scheduled_departure': scheduled_departure,
            'actual_departure': actual_departure
        }

        flights_data.append(flight_dict)

    return flights_data


# USE WITH FUNCTION FROM 1.1
root = load_xml("flights.xml")
if root is not None:
    flights_data = extract_flights(root)

    # Print statistics
    print(f"\nTotal flights: {len(flights_data)}")
    print("\nFirst 3 flights:")
    for i, flight in enumerate(flights_data[:3], 1):
        print(f"\n{i}. {flight}")
```

Expected output:

```
Total flights: 40

First 3 flights:
1. {'date': '2024-01-15', 'origin_id': 'DUB', 'destination_id': 'FCO',
    'aircraft_model': 'A320-214', 'scheduled_departure': '15:35',
    'actual_departure': '16:07'}

2. {'date': '2024-01-15', 'origin_id': 'DUB', 'destination_id': 'FCO',
    'aircraft_model': 'A320-214', 'scheduled_departure': '15:35',
    'actual_departure': '16:28'}

3. {'date': '2024-01-15', 'origin_id': 'DUB', 'destination_id': 'FCO',
    'aircraft_model': 'A320-214', 'scheduled_departure': '15:35',
    'actual_departure': '16:10'}
```

Question 1.3: Filter and Export Delayed Flights to CSV (10 marks)

Find flights that meet BOTH conditions: 1. Aircraft model is exactly `"A320-214"` 2. Actual departure time is LATER than scheduled departure time

Export to CSV file named `delayed_flights.csv`

Key insight: You can compare time strings directly if they're in HH:MM format - "16:07" > "15:35" = True (delayed)

Solution:

```python
import csv

def export_delayed_flights(flights_data, output_file="delayed_flights.csv"):
    """
    Filter delayed A320-214 flights and export to CSV.

    Args:
        flights_data (list): List of flight dictionaries from task 1.2
        output_file (str): Name of output CSV file
    """
    # Filter flights: A320-214 AND actual > scheduled
    delayed_flights = [
        flight for flight in flights_data
        if flight['aircraft_model'] == 'A320-214'
        and flight['actual_departure'] > flight['scheduled_departure']
    ]

    # Write to CSV
    with open(output_file, 'w', newline='', encoding='utf-8') as csvfile:
        # Define column headers
        fieldnames = [
            'date',
            'origin_id',
            'destination_id',
            'aircraft_model',
            'scheduled_departure',
            'actual_departure'
        ]

        # Create CSV writer
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

        # Write header row
        writer.writeheader()

        # Write data rows
        writer.writerows(delayed_flights)

    print(f"  Exported {len(delayed_flights)} delayed flights to {output_file}")


# USE WITH DATA FROM 1.2
export_delayed_flights(flights_data)
```

Check your output:

```bash
# Terminal: View the CSV file
cat delayed_flights.csv

# Expected: CSV with header row + data for delayed flights
```

Example CSV output:

```
date,origin_id,destination_id,aircraft_model,scheduled_departure,actual_departure
2024-01-15,DUB,FCO,A320-214,15:35,16:07
2024-01-15,DUB,FCO,A320-214,15:35,16:28
2024-01-15,DUB,FCO,A320-214,15:35,17:11
```

TASK 2: Lists, Dictionaries & NumPy (30 marks)

Dataset: Student records with scores

```python
records = [
    "2001,Ava,ENG,78;82;91",
    "2002,Noah,HIST,65;71;70",
    "2003,Mia,ENG,88;NA;92",
    "2004,Liam,CS,59;63;60",
    "2005,Zoe,ENG,NA;NA;84",
    "2006,Eli,HIST,74;70;69",
    "2007,Ivy,CS,90;95;93",
    "2008,Omar,ENG,72;68;70",
    "2009,Nia,HIST,55;58;61",
    "2010,Ruth,CS,81;79;85",
    "2011,Leo,ENG,64;67;NA",
    "2012,Uma,HIST,88;85;90",
]
```

Question 2.1: Convert to Structured Data (15 marks)

Convert `records` list into list of dictionaries using **list comprehensions**.

Requirements: - Keys: “id” (int), “name” (str), “subject” (str), “scores” (list of ints) - Ignore NA values in scores list - Use list comprehensions - No hard-coding beyond format specification - No external libraries except numpy (hint: for nanmean later)

Solution:

```
def parse_records(records):
    """
    Parse string records into structured dictionary format.

    Args:
        records (list): List of string records

    Returns:
        list: List of dictionaries with parsed data
    """

    parsed_data = [
        {
            'id': int(parts[0]),           # Convert student_id to int
            'name': parts[1],               # Name stays string
            'subject': parts[2],            # Subject stays string
            'scores': [
                int(score)                  # Convert each score to int
                for score in parts[3].split(';')  # Split scores by semicolon
                if score != 'NA'            # Skip NA values
            ]
        }
        for parts in (record.split(',') for record in records)  # Split each record by comm
    ]

    return parsed_data


# TEST IT
records = [
    "2001,Ava,ENG,78;82;91",
    "2002,Noah,HIST,65;71;70",
    "2003,Mia,ENG,88;NA;92",
    "2005,Zoe,ENG,NA;NA;84",
]

student_data = parse_records(records)

print("Parsed data:")
for student in student_data:
    print(student)
```

Expected output:

```
Parsed data:
{'id': 2001, 'name': 'Ava', 'subject': 'ENG', 'scores': [78, 82, 91]}
{'id': 2002, 'name': 'Noah', 'subject': 'HIST', 'scores': [65, 71, 70]}
{'id': 2003, 'name': 'Mia', 'subject': 'ENG', 'scores': [88, 92]}
{'id': 2005, 'name': 'Zoe', 'subject': 'ENG', 'scores': [84]}
```

Key insight: Mia (2003) has scores [88, 92] because NA is skipped. Zoe has [84] because two NAs are skipped.

Question 2.2: NumPy Array Analysis (15 marks)

Using the ORIGINAL `records` list: - Build NumPy array S of shape (N, 3) with each student's 3 scores - Convert "NA" to `np.nan` - For each student, compute: - `avg_np`: Mean using `np.nanmean()`, rounded to 1 decimal (None if all 3 missing) - `n_np`: Count of non-missing scores using `np.isnan()` - Print: `name - avg=X.X, n=Y`

Solution:

```
import numpy as np

def analyze_with_numpy(records):
    """
    Analyze student scores using NumPy with NaN handling.

    Args:
        records (list): List of string records
    """

    # Step 1: Parse records and extract scores
    N = len(records)
    S = np.zeros((N, 3))  # Initialize array of shape (N, 3)
    names = []

    for i, record in enumerate(records):
        parts = record.split(',')
        name = parts[1]
        names.append(name)

        scores_str = parts[3].split(';')

        # Convert each score, NA becomes NaN
        for j, score_str in enumerate(scores_str):
            if score_str == 'NA':
                S[i, j] = np.nan
            else:
                S[i, j] = float(score_str)

    # Step 2: Calculate averages and counts
    print("\nStudent Score Analysis:")
    print("-" * 50)

    for i, name in enumerate(names):
        # Get row for this student
        row = S[i, :]

        # Calculate mean (ignoring NaN)
        avg_np = np.nanmean(row)

        # Count non-NaN values
        n_np = np.sum(~np.isnan(row))

        # Handle case where all 3 scores are missing
        if n_np == 0:
            avg_np = None
            print(f"{name} - avg=None, n={int(n_np)}")
        else:
            avg_np = round(avg_np, 1)
            print(f"{name} - avg={avg_np}, n={int(n_np)}")


# TEST WITH FULL DATASET
records = [
    "2001,Ava,ENG,78;82;91",
    "2002,Noah,HIST,65;71;70",
    "2003,Mia,ENG,88;NA;92",
    "2004,Liam,CS,59;63;60",
    "2005,Zoe,ENG,NA;NA;84",
    "2006,Eli,HIST,74;70;69",
    "2007,Ivy,CS,90;95;93",
    "2008,Omar,ENG,72;68;70",
    "2009,Nia,HIST,55;58;61",
    "2010,Ruth,CS,81;79;85",
    "2011,Leo,ENG,64;67;NA",
    "2012,Uma,HIST,88;85;90",
]

analyze_with_numpy(records)
```

Expected output:

```
Student Score Analysis:
--------------------------------------------------
Ava - avg=83.7, n=3
Noah - avg=68.7, n=3
Mia - avg=90.0, n=2
Liam - avg=60.7, n=3
Zoe - avg=84.0, n=1
Eli - avg=71.0, n=3
Ivy - avg=92.7, n=3
Omar - avg=70.0, n=3
Nia - avg=58.0, n=3
Ruth - avg=81.7, n=3
Leo - avg=65.5, n=2
Uma - avg=87.7, n=3
```

Key concepts: - `np.nanmean()` - calculates mean ignoring NaN values - `np.isnan()` - returns True where value is NaN - `~np.isnan()` - inverts to True where value is NOT NaN - `np.sum(~np.isnan(row))` - counts non-NaN values

---

TASK 3: Text Analytics with Regex (30 marks)

Dataset: `reviews.txt` - Customer product reviews

Each line format:

```
<review_id> | <rating> | <review_text>
```

Example:

```
R001 | 5 | Love the product! #quality #value
R002 | 3 | Delivery was slow... @support please check. #delivery
R003 | NA | Not sure yet, will update later. @friend
```

Question 3.1: Load Reviews Function (15 marks)

Write function `load_reviews(filename)` that: - Opens file (UTF-8 encoding) - Skips empty lines - Splits on ” | ” separator - Converts rating: “NA” → None, otherwise → int - Returns list of dicts with keys: “id”, “rating”, “text”

Solution:

```python
def load_reviews(filename):
    """
    Load reviews from file into structured format.

    Args:
        filename (str): Path to reviews text file

    Returns:
        list: List of review dictionaries
    """
    reviews = []

    # Open and read file
    with open(filename, 'r', encoding='utf-8') as f:
        for line in f:
            # Skip empty lines
            line = line.strip()
            if not line:
                continue

            # Split on " | " separator
            parts = line.split(' | ')

            if len(parts) == 3:
                review_id = parts[0]
                rating_str = parts[1]
                review_text = parts[2]

                # Convert rating
                if rating_str == 'NA':
                    rating = None
                else:
                    rating = int(rating_str)

                # Create dictionary
                review_dict = {
                    'id': review_id,
                    'rating': rating,
                    'text': review_text
                }

                reviews.append(review_dict)

    return reviews
```

```python
# TEST IT
reviews = load_reviews("reviews.txt")
print(f"Total reviews loaded: {len(reviews)}")
print("\nFirst 3 reviews:")
for review in reviews[:3]:
    print(review)
```

Expected output:

```
Total reviews loaded: 25

First 3 reviews:
{'id': 'R001', 'rating': 5, 'text': 'Love the product! #quality #value'}
{'id': 'R002', 'rating': 3, 'text': 'Delivery was slow... @support please check. #delivery'
{'id': 'R003', 'rating': None, 'text': 'Not sure yet, will update later. @friend'}
```

Question 3.2: Extract Hashtags and Mentions (15 marks)

Write function `extract_tags(text)` using regex to: - Find all hashtags: `#` followed by 1+ letters, digits, underscores - Find all mentions: `@` followed by 1+ letters, digits, underscores - Return tuple: (hashtags_list, mentions_list)

Regex patterns:

```python
# Hashtags: # followed by word characters
hashtag_pattern = r'#[\w]+'    # \w = letters, digits, underscore

# Mentions: @ followed by word characters
mention_pattern = r'@[\w]+'
```

Solution:

```python
import re

def extract_tags(text):
    """
    Extract hashtags and mentions from review text using regex.

    Args:
        text (str): Review text

    Returns:
        tuple: (list of hashtags, list of mentions)
    """

    # Pattern for hashtags: # followed by 1+ word characters
    hashtag_pattern = r'#\w+'  # Matches #quality, #delivery2024, etc.
    hashtags = re.findall(hashtag_pattern, text)

    # Pattern for mentions: @ followed by 1+ word characters
    mention_pattern = r'@\w+'  # Matches @support, @user_1, etc.
    mentions = re.findall(mention_pattern, text)

    return (hashtags, mentions)


# TEST WITH REVIEWS
def analyze_all_reviews(reviews):
    """
    Extract and analyze tags for all reviews.

    Args:
        reviews (list): List of review dictionaries
    """
    print("Review Tag Analysis:")
    print("=" * 70)

    all_hashtags = {}
    all_mentions = {}

    for review in reviews:
        review_id = review['id']
        text = review['text']

        # Extract tags
        hashtags, mentions = extract_tags(text)

        # Print results
        print(f"\n{review_id}:")
        print(f"  Hashtags: {hashtags if hashtags else 'None'}")
        print(f"  Mentions: {mentions if mentions else 'None'}")

        # Count occurrences for summary
        for tag in hashtags:
            all_hashtags[tag] = all_hashtags.get(tag, 0) + 1

        for mention in mentions:
            all_mentions[mention] = all_mentions.get(mention, 0) + 1

    # Print summary
    print("\n" + "=" * 70)
    print("\nMost Common Hashtags:")
    for tag, count in sorted(all_hashtags.items(), key=lambda x: x[1], reverse=True)[:5]:
        print(f"  {tag}: {count} times")

    print("\nMost Mentioned Users:")
    for mention, count in sorted(all_mentions.items(), key=lambda x: x[1], reverse=True)[:5]
        print(f"  {mention}: {count} times")


# TEST IT
reviews = load_reviews("reviews.txt")
analyze_all_reviews(reviews)
```

Expected output:

```
Review Tag Analysis:
======================================================================

R001:
  Hashtags: ['#quality', '#value']
  Mentions: []

R002:
  Hashtags: ['#delivery']
  Mentions: ['@support']

R003:
  Hashtags: []
  Mentions: ['@friend']

R004:
  Hashtags: ['#quality', '#packaging']
  Mentions: []

[... more reviews ...]

======================================================================

Most Common Hashtags:
  #quality: 5 times
  #delivery: 4 times
  #value: 3 times
  #packaging: 2 times
  #ux: 2 times

Most Mentioned Users:
  @support: 4 times
  @alice: 1 times
  @bob: 1 times
```

```
@friend: 1 times
@tester: 1 times
```

CA SET 2: A COHORT ASSESSMENT (100 marks total)

Overview

This assessment covers: - **Task 1**: XML File Handling (55 marks) - Working with reed.xml - **Task 2**: Array Manipulation (30 marks) - NumPy 3D arrays - **Task 3**: Regex Currency Extraction (15 marks) - Advanced regex

Data files provided: - `reed.xml` - Reed College course listings

TASK 1: XML File Handling with Reed College Courses (55 marks)

Dataset: `reed.xml` - Contains course records with: - Registration number, subject, course number, section - Title, units (credits), instructor - Meeting days, start/end times, building, room

Question 1.1: Safe XML Load Function (20 marks)

Write function `load_xml(path)` that: - Opens and parses XML safely - Handles: missing file, permission denied, malformed XML - Returns root element on success, None on failure - Display clear error messages

Solution:

```python
import xml.etree.ElementTree as ET

def load_xml(path):
    """
    Safely load and parse an XML file with comprehensive error handling.

    Args:
        path (str): Path to XML file

    Returns:
        ET.Element: Root element if successful, None if failed
    """
    try:
        # Parse XML file
        tree = ET.parse(path)
        root = tree.getroot()
        print(f" XML file '{path}' loaded successfully")
        return root

    except FileNotFoundError:
        print(f" Error: File '{path}' not found.")
        print(f"  Please check the file path and try again.")
        return None

    except PermissionError:
        print(f" Error: Permission denied when reading '{path}'.")
        print(f"  Check file permissions.")
        return None

    except ET.ParseError as e:
        print(f" Error: XML parsing failed.")
        print(f"  Details: {e}")
        return None

    except IOError as e:
        print(f" Error: I/O error reading file.")
        print(f"  Details: {e}")
        return None

    except Exception as e:
        print(f" Unexpected error: {e}")
        return None


# TEST IT
root = load_xml("reed.xml")
```

Question 1.2: Select and Display Specific Courses (15 marks)

Using the parsed XML, output courses at positions 3, 7, 11, 15, 19 (document order):

For each course, print: - Registration number, subject, course number, section, title - Meeting days and raw start/end times

Handle missing indices gracefully.

Solution:

```python
def display_selected_courses(root, indices=[3, 7, 11, 15, 19]):
    """
    Display specific courses by index with complete information.

    Args:
        root (ET.Element): Root element from XML
        indices (list): List of course indices to display (1-based)
    """
    # Get all course elements
    courses = root.findall('.//course')
    total_courses = len(courses)

    print(f"Total courses in dataset: {total_courses}")
    print("=" * 100)

    missing_count = 0

    for idx in indices:
        # Convert to 0-based index
        course_idx = idx - 1

        # Check if index exists
        if course_idx >= total_courses:
            missing_count += 1
            continue

        # Get course element
        course = courses[course_idx]

        # Extract required fields
        reg_number = course.findtext('registration_number', 'N/A')
        subject = course.findtext('subject', 'N/A')
        course_number = course.findtext('course_number', 'N/A')
        section = course.findtext('section', 'N/A')
        title = course.findtext('title', 'N/A')

        days = course.findtext('meeting_days', 'N/A')
        start_time = course.findtext('start_time', 'N/A')
        end_time = course.findtext('end_time', 'N/A')

        # Print formatted output
        print(f"\nCourse #{idx}:")
        print(f"  Reg#: {reg_number} | {subject} {course_number}-{section}")
        print(f"  Title: {title}")
        print(f"  Days: {days} | Time: {start_time} - {end_time}")
    # Print warning if courses missing
    if missing_count > 0:
        print("\n" + "=" * 100)
        print(f"  Warning: {missing_count} requested course(s) do not exist (only {total_cou


# TEST IT
root = load_xml("reed.xml")
if root is not None:
    display_selected_courses(root)
```

Expected output format:

```
Total courses in dataset: 500
===========================================================================================

Course #3:
  Reg#: R003 | ENG 101-A
  Title: Introduction to Literature
  Days: MWF | Time: 09:00 - 10:00

Course #7:
  Reg#: R007 | HIST 200-B
  Title: American History
  Days: TR | Time: 13:00 - 14:30

[... more courses ...]

===========================================================================================
  Warning: 0 requested course(s) do not exist (only 500 available)
```

Question 1.3: Filter and Export ENG/HIST Full-Credit Courses (20 marks)

Filter courses matching ALL criteria: 1. Subject is either “ENG” or “HIST” 2. Units = 1.0 (full credit) 3. Instructor is non-empty (after trimming) 4. Building is non-empty (after trimming)

Export to CSV: `eng_hist_fullcredit.csv` with columns:

```
subj, crse, sect, title, instructor, days, start_time, end_time, building, room
```

Solution:

```
import csv

def export_eng_hist_courses(root, output_file="eng_hist_fullcredit.csv"):
```

```python
"""
Filter and export ENG/HIST full-credit courses to CSV.

Args:
    root (ET.Element): Root element from XML
    output_file (str): Output CSV filename
"""

# Get all courses
courses = root.findall('.//course')

# Filter based on criteria
filtered_courses = []

for course in courses:
    # Get fields
    subject = course.findtext('subject', '').strip()
    units = course.findtext('units', '0')
    instructor = course.findtext('instructor', '').strip()
    building = course.findtext('building', '').strip()

    # Check all criteria
    # 1. Subject is ENG or HIST
    if subject not in ['ENG', 'HIST']:
        continue

    # 2. Units = 1.0 (full credit)
    try:
        units_float = float(units)
        if units_float != 1.0:
            continue
    except ValueError:
        continue

    # 3. Instructor non-empty
    if not instructor:
        continue

    # 4. Building non-empty
    if not building:
        continue

    # All criteria met - extract data
    course_number = course.findtext('course_number', 'N/A')
    section = course.findtext('section', 'N/A')
    title = course.findtext('title', 'N/A')
```

```
        days = course.findtext('meeting_days', 'N/A')
        start_time = course.findtext('start_time', 'N/A')
        end_time = course.findtext('end_time', 'N/A')
        room = course.findtext('room', 'N/A')

        # Build row dictionary
        row = {
            'subj': subject,
            'crse': course_number,
            'sect': section,
            'title': title,
            'instructor': instructor,
            'days': days,
            'start_time': start_time,
            'end_time': end_time,
            'building': building,
            'room': room
        }

        filtered_courses.append(row)

    # Write to CSV
    if filtered_courses:
        fieldnames = [
            'subj', 'crse', 'sect', 'title', 'instructor',
            'days', 'start_time', 'end_time', 'building', 'room'
        ]

        with open(output_file, 'w', newline='', encoding='utf-8') as csvfile:
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            writer.writerows(filtered_courses)

        print(f"  Exported {len(filtered_courses)} courses to {output_file}")
    else:
        print("  No courses matched the filtering criteria")


# TEST IT
root = load_xml("reed.xml")
if root is not None:
    export_eng_hist_courses(root)
```

TASK 2: 3D Array Manipulation (30 marks)

Complete NumPy array operations without loops where possible.

Question 2.1: Create 3D Array (7 marks)

Create array of shape (9, 12, 12) where element [k, i, j] = (k+1)*(i+1) + (j+1)

Solution:

```
import numpy as np

# Create 3D array using vectorized operations
# Shape: (9, 12, 12)

k_indices = np.arange(9)  # 0 to 8
i_indices = np.arange(12)  # 0 to 11
j_indices = np.arange(12)  # 0 to 11

# Reshape for broadcasting
k = k_indices[:, np.newaxis, np.newaxis]  # Shape (9, 1, 1)
i = i_indices[np.newaxis, :, np.newaxis]  # Shape (1, 12, 1)
j = j_indices[np.newaxis, np.newaxis, :]  # Shape (1, 1, 12)

# Create array using formula: (k+1)*(i+1) + (j+1)
arrA = (k + 1) * (i + 1) + (j + 1)

print(f"Array shape: {arrA.shape}")
print(f"Element [0, 0, 0] = {arrA[0, 0, 0]}")  # Should be (0+1)*(0+1) + (0+1) = 2
print(f"Element [1, 2, 3] = {arrA[1, 2, 3]}")  # Should be (1+1)*(2+1) + (3+1) = 10
```

Expected output:

```
Array shape: (9, 12, 12)
Element [0, 0, 0] = 2
Element [1, 2, 3] = 10
```

Question 2.2: Replace Multiples of 7 with Sentinel (7 marks)

Replace every element divisible by 7 with -9999. Return 1D array showing count per layer (9 values).

Solution:

```
def replace_multiples_of_7(arrA):
    """
    Replace elements divisible by 7 with -9999.
    Return count per layer.
    Args:
        arrA (np.ndarray): 3D array

    Returns:
        np.ndarray: 1D array with replacement count per layer
    """

    # Copy to avoid modifying original
    arr_modified = arrA.copy()

    # Find elements divisible by 7 (remainder 0 when divided by 7)
    divisible_by_7 = (arr_modified % 7 == 0)

    # Replace with sentinel
    arr_modified[divisible_by_7] = -9999

    # Count replacements per layer
    count_per_layer = np.zeros(arrA.shape[0], dtype=int)

    for k in range(arrA.shape[0]):
        # Count -9999 values in this layer
        count_per_layer[k] = np.sum(arr_modified[k, :, :] == -9999)

    print(f"Replacements per layer: {count_per_layer}")
    print(f"Total replacements: {np.sum(count_per_layer)}")

    return arr_modified, count_per_layer


# TEST IT
arrA_modified, replacements = replace_multiples_of_7(arrA)
```

Question 2.3: Sum Specific Layers (6 marks)

Compute sum of all elements in layers 2-5 (0-based, inclusive). Store in variable `layer_sum`.

Solution:

```
def sum_layers(arrA, layer_modified, start_layer=2, end_layer=5):
    """
    Sum elements in specific layers (ignoring sentinel values).

    Args:
        arrA (np.ndarray): Original array
        layer_modified (np.ndarray): Modified array with sentinels
        start_layer (int): Start layer index (0-based)
        end_layer (int): End layer index (0-based, inclusive)

    Returns:
        float: Sum of elements in specified layers (excluding sentinels)
    """

    # Extract layers 2-5
    layers = layer_modified[start_layer:end_layer+1, :, :]

    # Sum all elements except sentinel values
    layer_sum = np.sum(layers[layers != -9999])

    print(f"Sum of layers {start_layer}-{end_layer}: {layer_sum}")
    return layer_sum


# TEST IT
layer_sum = sum_layers(arrA, arrA_modified)
```

Question 2.4: Broadcasting Column Scaling (10 marks)

Create 1D array of scaling factors (1 to N, where N = number of columns). Multiply arrA by this scaling vector using broadcasting. Print mean of result (excluding sentinels).

Solution:

```
def apply_column_scaling(arrA):
    """
    Scale each column by column index + 1.

    Args:
        arrA (np.ndarray): 3D array to scale

    Returns:
        np.ndarray: Scaled array
    """

    # Number of columns
    N = arrA.shape[2]  # Last dimension

    # Create scaling factors: 1 to N
    scaling_factors = np.arange(1, N + 1)  # [1, 2, 3, ..., N]

    print(f"Scaling factors: {scaling_factors}")

    # Reshape for broadcasting
    # arrA shape: (9, 12, 12)
    # scaling_factors shape: (12,) -> reshape to (1, 1, 12) for broadcasting
    scaling_vector = scaling_factors[np.newaxis, np.newaxis, :]

    # Apply scaling
    scaled_array = arrA * scaling_vector

    # Calculate mean (excluding sentinel values -9999)
    valid_elements = scaled_array[scaled_array != -9999]
    mean_value = np.mean(valid_elements)

    print(f"Mean of scaled array (excluding sentinels): {mean_value:.2f}")

    return scaled_array


# TEST IT
scaled_arr = apply_column_scaling(arrA_modified)
```

TASK 3: Extract and Normalise Currency Amounts with Regex (15 marks)

Text may contain currency in formats: - €1,234.50, EUR 99, $12.3, USD 1,000, €5, USD12.00

Write function `extract_money(text)` that: - Finds currency markers: €, EUR, $, USD (case-insensitive) - Handles optional thousands separators and decimals - Normalises to: “CUR amount” format (e.g., “EUR 1234.50”, “USD 12.00”) - Returns: (normalised_list, eur_count, usd_count)

Solution:

```
import re

def extract_money(text):
    """
    Extract and normalise currency amounts from text.

    Args:
        text (str): Input text with currency amounts
    Returns:
        tuple: (normalised_list, eur_count, usd_count)
    """

    # Regex pattern to match currency amounts
    # Matches: € or EUR or $ or USD, followed by optional spaces and amount
    # Amount can have: digits, commas, decimal points
    pattern = r'(€|EUR|US|USD)\s*([0-9,]+(?:\.[0-9]{1,2})?)'

    matches = re.finditer(pattern, text, re.IGNORECASE)

    normalised_list = []
    eur_count = 0
    usd_count = 0

    for match in matches:
        currency_marker = match.group(1).upper()
        amount_str = match.group(2)

        # Normalize currency: € or EUR -> EUR, $ or USD -> USD
        if currency_marker in ['€', 'EUR']:
            currency = 'EUR'
            eur_count += 1
        elif currency_marker in ['$', 'USD']:
            currency = 'USD'
            usd_count += 1

        # Clean amount: remove commas
        amount_clean = amount_str.replace(',', '')

        # Convert to float and format to 2 decimals
        try:
            amount_float = float(amount_clean)
            amount_formatted = f"{amount_float:.2f}"
        except ValueError:
            continue

        # Normalize format
        normalized = f"{currency} {amount_formatted}"
        normalised_list.append(normalized)

    return (normalised_list, eur_count, usd_count)


# TEST WITH SAMPLE TEXT
test_text = """
```

```
Order summary:
Laptop €1,099, Mouse EUR45 .5, and USD 2 ,000 for travel.
Refund: $50 , discount €25.00 , bonus USD10 .75.
Ignored text: price = 100 (no currency), Paid: eur 3 ,250.5 and usd1 ,500.
"""

result, eur_cnt, usd_cnt = extract_money(test_text)

print(f"Normalised amounts: {result}")
print(f"EUR count: {eur_cnt}")
print(f"USD count: {usd_cnt}")
```

Expected output:

```
Normalised amounts: ['EUR 1099.00', 'EUR 45.50', 'USD 2000.00', 'USD 50.00', 'EUR 25.00', 'U
EUR count: 4
USD count: 4
```

ADDITIONAL PRACTICE QUESTIONS

These are bonus questions similar in difficulty and topics to the official CA assessments.

BONUS TASK 1: Hotel Booking System

Scenario: You have XML data for hotel bookings with guest information, room types, check-in/out dates.

Bonus 1.1: Safe XML Loading

Write `load_hotel_xml(path)` with exception handling.

Bonus 1.2: Calculate Average Stay Duration

Extract check-in and check-out dates, calculate stay in days for each booking.

Bonus 1.3: Filter Premium Rooms

Filter rooms where price > €200/night AND rating > 4.0, export to CSV.

BONUS TASK 2: Student Performance Analysis

Scenario: Similar to CA Task 2, but with quarterly grades.

```
quarterly_records = [
    "S001,Alice,Q1:85;88;92,Q2:80;85;90",
    "S002,Bob,Q1:NA;75;78,Q2:82;85;79",
    "S003,Charlie,Q1:88;90;92,Q2:NA;NA;NA",
]
```

Bonus 2.1: Parse Quarterly Data

Convert to nested dict: `{student_id: {quarter: [scores]}}`

Bonus 2.2: Compare Quarters

Calculate if Q2 average > Q1 average for each student.

Bonus 2.3: Create Summary Report

Print ranking by average score, identify improving students.

BONUS TASK 3: Social Media Analytics

Scenario: CSV log file with timestamps, user IDs, engagement metrics.

Bonus 3.1: Time-Based Regex Extraction

Extract timestamps in format HH:MM:SS and convert to datetime.

Bonus 3.2: Mention Thread Analysis

Track @mention threads - when User A mentions User B, who responds.

Bonus 3.3: Hashtag Trending

Find hashtags appearing in >5 posts, calculate popularity trend.

COMPLETE CODE TEMPLATES

Template 1: XML File Handling Pattern

```
import xml.etree.ElementTree as ET
import csv
```

```
class XMLProcessor:
    """Reusable XML processing with error handling"""

    @staticmethod
    def load_safe(filepath):
        try:
            tree = ET.parse(filepath)
            return tree.getroot()
        except FileNotFoundError:
            print(f"  File not found: {filepath}")
            return None
        except ET.ParseError as e:
            print(f"  XML Parse Error: {e}")
            return None

    @staticmethod
    def extract_records(root, record_tag):
        """Generic record extraction"""
        return root.findall(f'.//{record_tag}')

    @staticmethod
    def export_to_csv(data, filepath, fieldnames):
        """Generic CSV export"""
        with open(filepath, 'w', newline='', encoding='utf-8') as f:
            writer = csv.DictWriter(f, fieldnames=fieldnames)
            writer.writeheader()
            writer.writerows(data)
        print(f"  Exported {len(data)} records to {filepath}")
```

Template 2: NumPy Data Analysis Pattern

```
import numpy as np

class DataAnalyzer:
    """NumPy-based data analysis patterns"""

    @staticmethod
    def handle_missing(data):
        """Convert 'NA' to np.nan"""
        return np.array([np.nan if x == 'NA' else float(x) for x in data])

    @staticmethod
    def calc_stats(array):
        """Calculate stats ignoring NaN"""
        return {
            'mean': np.nanmean(array),
            'median': np.nanmedian(array),
            'std': np.nanstd(array),
            'count': np.sum(~np.isnan(array))
        }

    @staticmethod
    def array_operations(arr):
        """Vectorized array operations"""
        # Replace values divisible by 7
        arr[arr % 7 == 0] = -9999

        # Count replacements
        replacements = np.sum(arr == -9999)

        return arr, replacements
```

Template 3: Regex Pattern Collection

```
import re

class RegexPatterns:
    """Common regex patterns for data extraction"""

    # Email validation
    EMAIL = r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'

    # Hashtags
    HASHTAG = r'#[\w]+'

    # Mentions
    MENTION = r'@[\w]+'

    # Currency (€, EUR, $, USD)
    CURRENCY = r'(€|EUR|\$|USD)\s*([0-9,]+(?:\.[0-9]{1,2})?)'

    # Phone numbers (US format)
    PHONE_US = r'\(\d{3}\)\s?\d{3}-\d{4}'

    # URLs
    URL = r'https?://[\w./\-]+'
    # ISO Date (YYYY-MM-DD)
    DATE_ISO = r'\d{4}-\d{2}-\d{2}'

    # Time (HH:MM or HH:MM:SS)
    TIME = r'\d{2}:\d{2}(?::\d{2})?'

    @staticmethod
    def extract(pattern, text, flags=0):
        """Generic extraction"""
        return re.findall(pattern, text, flags)

    @staticmethod
    def replace(pattern, replacement, text):
        """Generic replacement"""
        return re.sub(pattern, replacement, text)
```

QUICK ANSWER CHECKLIST

Use this to verify your solutions:

Task 1: XML Handling

- □ Function handles FileNotFoundError
- □ Function handles PermissionError
- □ Function handles ParseError
- □ Returns None on error, root on success
- □ All attributes extracted with `.get()`
- □ All text extracted with `.findtext()`
- □ List built with correct keys
- □ CSV written with DictWriter
- □ Filtered correctly (all conditions checked)

Task 2: Lists & Dictionaries

- □ List comprehensions used
- □ NA values skipped (not included)
- □ Dictionary keys spelled exactly as specified
- □ Scores parsed as integers
- □ IDs converted to integers

Task 2: NumPy Analysis

- □ 3D array created with correct shape

☐ Formula applied correctly
☐ Element replacement works
☐ Count per layer computed
☐ Layer sum ignores sentinels
☐ Broadcasting applies column scaling
☐ Means computed excluding -9999

Task 3: Regex & Text

☐ File read line by line
☐ Split on " | " (with spaces)
☐ NA converts to None
☐ Other ratings convert to int
☐ Hashtag pattern finds all tags
☐ Mention pattern finds all mentions
☐ Case-insensitive matching used
☐ Currency patterns work with spaces
☐ Amounts formatted to 2 decimals
☐ Comma removal works

TROUBLESHOOTING GUIDE

XML Issues

Problem: ElementTree not found

```
# Solution
import xml.etree.ElementTree as ET
```

Problem: Getting None when accessing attributes

```
# Wrong
flight['origin_id']  # Lists don't have string keys!

# Right
flight_dict = {'origin_id': 'DUB'}
print(flight_dict['origin_id'])
```

Problem: Attributes not being extracted

```
# Wrong - attributes are accessed differently
origin_elem.findtext('id')

# Right
```

```
origin_elem = flight.find('origin')
origin_id = origin_elem.get('id')
```

NumPy Issues

Problem: Shape mismatch in broadcasting

```
# Check shapes
print(f"arrA shape: {arrA.shape}")  # (9, 12, 12)
print(f"scaling shape: {scaling_vector.shape}")  # Must be (1, 1, 12)

# Fix: Add newaxis
scaling_vector = scaling_factors[np.newaxis, np.newaxis, :]
```

Problem: NaN propagation in calculations

```
# Wrong - NaN makes entire result NaN
result = array1 + array2  # If any value is NaN, result has NaN

# Right - use nanmean, nansum, etc.
mean_value = np.nanmean(array1)
```

Regex Issues

Problem: Pattern not matching

```
# Check for:
# 1. Missing flags
pattern = r'...'
matches = re.findall(pattern, text, re.IGNORECASE)  # Add flags!

# 2. Missing escape characters
# Wrong
price = r'$100'  # $ has special meaning

# Right
price = r'\$100'  # Escape it
```

Problem: Groups not capturing

```
# Wrong
text = "price: EUR100"
pattern = r'(EUR)(\d+)'
result = re.findall(pattern, text)  # Returns list of tuples!

# Right
```

```
result = re.findall(pattern, text)  # [('EUR', '100')]
currency, amount = result[0]
```

FINAL TIPS FOR SUCCESS

1. **Test incrementally** - Don't write all code at once

   ```
   # Test loading first
   root = load_xml("data.xml")
   if root is not None:
       # Then test extraction
   ```

2. **Print intermediate results** - Verify data as you process

   ```
   print(f"Loaded {len(flights_data)} flights")
   print(f"First flight: {flights_data[0]}")
   ```

3. **Handle edge cases** - Empty lists, missing files, NA values

   ```
   if not data:
       print("No data to process")
   ```

4. **Use meaningful variable names** - Not x, y, z

   ```
   # Good
   flight_name = flight.findtext('name')
   departure_time = flight.findtext('scheduled_departure')

   # Bad
   f = flight.findtext('name')
   dt = flight.findtext('scheduled_departure')
   ```

5. **Comment your regex patterns** - Explain what they match

   ```
   # Match email: name@domain.com
   EMAIL_PATTERN = r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'
   ```

6. **Verify CSV output** - Check file exists and has correct structure

   ```
   # Terminal
   head -3 delayed_flights.csv  # Show first 3 lines
   wc -l delayed_flights.csv    # Count lines
   ```

Good luck with your assessments! You've got this!