Table of Contents:

Task 1. Design, build, test, explain, adapt, critique (MLO1, MLO2)

1. **(a) Implementation of Basic Game Mechanics**
1. **(b) Design, Development, and Testing Explanation**
1. **(c) Critical Enhancement with Shot Clock and Performance Analysis**

Task 2. Critically assess, select and apply data science libraries and algorithms (MLO3)

2. **(a) Problem Definition and Manual Analysis**
2. **(b) Automated Implementation Using NLTK and Pandas**
2. **(c) Critical Evaluation of Libraries**

Task 1. Design, build, test, explain, adapt, critique (MLO1, MLO2)

1. (a) Implementation of Basic Game Mechanics

CODE:

```
"""
Soccer Dice: Task 1A Implementation

Simulation of a football/soccer match between two teams.

"""

import random as rand


def roll_die(die_name, die_faces):

    roll_index = rand.randrange(len(die_faces))
    outcome = die_faces[roll_index]
    print("    Rolled " + die_name + " die: '" + outcome + "'")
    return outcome, roll_index


def play_possession(attacking_team, defending_team, current_score):

    """
    Simulate one complete attacking possession for attacking_team.

    """

    # Defining the dice faces (unedited)
    red = ['through ball/to yellow', 'inside pass/to blue', 'dribble/throw again',
           'short pass/to black', 'through ball/to green', 'tackled and lost']
    black = ['pass back/to red', 'throw in/to blue', 'shoot',
             'free kick/to yellow', 'long shot at goal', 'tackled and lost']
    blue = ['header at goal', 'shoot', 'opponent shown yellow card',
            'pass back/to red', 'long shot at goal', 'tackled and lost']
    yellow = ['pass back/to black', 'off side', 'tackled and lost',
              'penalty', 'shoot', 'shoot']
    green = ['goal', 'wide', 'goal', 'over bar', 'saved', 'corner/to yellow']

    # Prints the possession header with current score
    print("\n" + attacking_team + " team attacks (current score: Home " +
          str(current_score[0]) + "-" + str(current_score[1]) + " Away)")
    print("  Starting on Red die...")

    # To initialize the possession state
    next_die = 'Red'
    keep_going = True

    while keep_going:

        # Rolling the appropriate die based on current state
        if next_die == 'Red':
            outcome, idx = roll_die('Red', red)

            if outcome == 'tackled and lost':
                print("    -> " + attacking_team + " loses possession. " +
```

```
                    defending_team + " will attack next.")
                keep_going = False
                return current_score, defending_team

            elif outcome == 'through ball/to yellow':
                print("    -> Amazing through ball, switching to Yellow")
                next_die = 'Yellow'

            elif outcome == 'inside pass/to blue':
                print("    -> Neat pass, switching to Blue")
                next_die = 'Blue'

            elif outcome == 'dribble/throw again':
                print("    -> Skilful dribble, rolling Red again")
                next_die = 'Red'

            elif outcome == 'short pass/to black':
                print("    -> Short possession-retaining pass to Black")
                next_die = 'Black'

            elif outcome == 'through ball/to green':
                print("    -> Great pass! Straight to Green for a shot")
                next_die = 'Green'

        elif next_die == 'Black':
            outcome, idx = roll_die('Black', black)

            if outcome == 'tackled and lost':
                print("    -> Strong tackle! " + defending_team + " regains
possession")
                keep_going = False
                return current_score, defending_team

            elif outcome == 'pass back/to red':
                print("    -> Recycling play, passing back to Red")
                next_die = 'Red'

            elif outcome == 'throw in/to blue':
                print("    -> Throw-in down the line, switching to Blue")
                next_die = 'Blue'

            elif outcome == 'shoot':
                print("    -> Space opens up! Taking a shot — switching to Green")
                next_die = 'Green'

            elif outcome == 'free kick/to yellow':
                print("    -> Foul! Free kick opportunity to Yellow")
                next_die = 'Yellow'

            elif outcome == 'long shot at goal':
                print("    -> Long-shot opportunity! Rolling Green for the attempt")
                next_die = 'Green'

        elif next_die == 'Blue':
            outcome, idx = roll_die('Blue', blue)

            if outcome == 'tackled and lost':
                print("    -> Dispossessed! " + defending_team + " has the ball")
                keep_going = False
                return current_score, defending_team
```

```
            elif outcome == 'header at goal':
                print("    -> Head towards goal — to Green")
                next_die = 'Green'

            elif outcome == 'shoot':
                print("    -> Clear sight of goal! Taking a shot — to Green")
                next_die = 'Green'

            elif outcome == 'opponent shown yellow card':
                print("    -> Opponent cautioned! " + attacking_team + " gets a free
shot to Green")
                next_die = 'Green'

            elif outcome == 'pass back/to red':
                print("    -> Recycles the ball back to Red")
                next_die = 'Red'

            elif outcome == 'long shot at goal':
                print("    -> Attempt from distance towards Green")
                next_die = 'Green'

        elif next_die == 'Yellow':
            outcome, idx = roll_die('Yellow', yellow)

            if outcome == 'off side':
                print("    -> OFFSIDE! Attack ends")
                keep_going = False
                return current_score, defending_team

            elif outcome == 'tackled and lost':
                print("    -> Possession lost in midfield")
                keep_going = False
                return current_score, defending_team

            elif outcome == 'pass back/to black':
                print("    -> Sideways pass to Black")
                next_die = 'Black'

            elif outcome == 'penalty':
                print("    -> Penalty awarded! Direct opportunity to Green")
                next_die = 'Green'

            elif outcome == 'shoot':
                print("    -> Shooting opportunity! Rolling Green")
                next_die = 'Green'

        elif next_die == 'Green':
            outcome, idx = roll_die('Green', green)

            if outcome == 'goal' or outcome == 'goal':
                print("    -> *** GOAL! ***")

                # Update the score
                if attacking_team == 'Home':
                    current_score = (current_score[0] + 1, current_score[1])
                else:
                    current_score = (current_score[0], current_score[1] + 1)

                print("    -> Score updated: Home " + str(current_score[0]) +
                      " - " + str(current_score[1]) + " Away")
                print("    -> Kick-off to " + defending_team)
```

```
                keep_going = False
                return current_score, defending_team

            elif outcome == 'wide':
                print("    -> Shot goes wide! Goal kick — attack ends")
                keep_going = False
                return current_score, defending_team

            elif outcome == 'over bar':
                print("    -> Over the bar! Goal kick — attack ends")
                keep_going = False
                return current_score, defending_team

            elif outcome == 'saved':
                print("    -> Goalkeeper makes a save! Possession ends")
                keep_going = False
                return current_score, defending_team

            elif outcome == 'corner/to yellow':
                print("    -> Deflection off the goalkeeper — corner awarded!")
                print("    -> Corner kick back to Yellow")
                next_die = 'Yellow'


def play_match(seed, num_attacks):

    #Orchestrate a complete match between Home and Away teams.

    rand.seed(seed)
    score = (0, 0)
    current_attacker = 'Home'
    current_defender = 'Away'

    print("=" * 69)
    print("SOCCER DICE MATCH SIMULATION")
    print("=" * 69)
    print("Kick-off: Home team attacks first")
    print("Simulating " + str(num_attacks) + " attacking possessions")
    print("=" * 69)

    for possession_number in range(1, num_attacks + 1):
        print("\n[Possession " + str(possession_number) + "]")
        score, next_attacker = play_possession(current_attacker, current_defender,
score)
        current_attacker, current_defender = next_attacker, current_attacker

    print("\n" + "=" * 69)
    print("FULL-TIME WHISTLE")
    print("=" * 69)
    print("FINAL SCORE: Home " + str(score[0]) + " - " + str(score[1]) + " Away")
    print("=" * 69 + "\n")


# EXECUTION OF THE MATCH
play_match(seed=23, num_attacks=5)
```

OUTPUT:

```
==========================================================================
SOCCER DICE MATCH SIMULATION
==========================================================================
Kick-off: Home team attacks first
Simulating 5 attacking possessions
==========================================================================

[Possession 1]

Home team attacks (current score: Home 0-0 Away)
  Starting on Red die...
    Rolled Red die: 'dribble/throw again'
    -> Skilful dribble, rolling Red again
    Rolled Red die: 'through ball/to yellow'
    -> Amazing through ball, switching to Yellow
    Rolled Yellow die: 'pass back/to black'
    -> Sideways pass to Black
    Rolled Black die: 'long shot at goal'
    -> Long-shot opportunity! Rolling Green for the attempt
    Rolled Green die: 'goal'
    -> *** GOAL! ***
    -> Score updated: Home 1 - 0 Away
    -> Kick-off to Away


[Possession 2]

Away team attacks (current score: Home 1-0 Away)
  Starting on Red die...
    Rolled Red die: 'short pass/to black'
    -> Short possession-retaining pass to Black
    Rolled Black die: 'free kick/to yellow'
    -> Foul! Free kick opportunity to Yellow
    Rolled Yellow die: 'shoot'
    -> Shooting opportunity! Rolling Green
    Rolled Green die: 'goal'
    -> *** GOAL! ***
    -> Score updated: Home 1 - 1 Away
    -> Kick-off to Home

[Possession 3]

Home team attacks (current score: Home 1-1 Away)
  Starting on Red die...
    Rolled Red die: 'inside pass/to blue'
    -> Neat pass, switching to Blue
    Rolled Blue die: 'tackled and lost'
    -> Dispossessed! Away has the ball
```

[Possession 4]

Away team attacks (current score: Home 1-1 Away)
  Starting on Red die...
    Rolled Red die: 'inside pass/to blue'
    -> Neat pass, switching to Blue
    Rolled Blue die: 'opponent shown yellow card'
    -> Opponent cautioned! Away gets a free shot to Green
    Rolled Green die: 'over bar'
    -> Over the bar! Goal kick — attack ends

[Possession 5]

Home team attacks (current score: Home 1-1 Away)
  Starting on Red die...
    Rolled Red die: 'through ball/to yellow'
    -> Amazing through ball, switching to Yellow
    Rolled Yellow die: 'off side'
    -> OFFSIDE! Attack ends

==========
FULL-TIME WHISTLE
==========
FINAL SCORE: Home 1 - 1 Away
==========

1. **(b) Design, Development, and Testing Explanation**

Implementation of My Soccer Dice:

It utilizes three key functions: rolldie(), playpossession(), and playmatch(). rolldie() is used to create the random dice values for the game, while playpossession() is responsible for creating the game state machine (transitioning between Red → Black → Blue → Yellow → Green), and playmatch() is utilized to create a series of possessions and implement an alternating teams feature. Each function is designed with a single responsibility, and parameters are used to pass information between functions to ensure that they have a high level of cohesion, and a low level of coupling.

The main assumptions that were made during the design of this program, include:

- Possessions will always start as a result of rolling the Red die.
- Possessions will end due to one of the following conditions: being tackled or losing possession (Yellow), going out of bounds or offside (Yellow), or reaching a green outcome (except for a corner, which is returned to Yellow).
- Scores will be represented as a tuple and passed from function to function.
- random.seed() was utilized to provide a predictable environment for the testing of the program.

Development Stages:

The development of the soccer dice application was accomplished in five incremental stages.

Stage 1 - Die Rolling: The rolldie() function was developed and tested with a fixed seed value to confirm that the six possible outcomes (Red, Black, Blue, White, Green, and Yellow) can occur.

Stage 2 - Red Die Logic: The Red die transitions were implemented into the playpossession() function using an if/elif chain, and the transitions between Red and the other colors (Yellow, Blue, Black, and Green) were tested manually by setting the seed to 42.

Stage 3 - Additional Dice: The additional die logic for Black, Blue, Yellow, and Green were progressively implemented and tested after each stage.

Stage 4 - Score Tracking: Conditional score updates were implemented based upon whether the Green die resulted in a "goal". The score tracking functionality was then tested by comparing it to the number of goals scored manually.

Stage 5 - Match Orchestration: The playmatch() function was implemented with the ability to alternate teams and control the seed value.

Bugs:

A total of four bugs were discovered and corrected during the development of the soccer dice application. These bugs included:

Missing elif branch (Stage 2): If the current die was not the same as the next die, the next die would not be updated. This bug was fixed by adding missing elif branches.

Incorrect Team Scoring (Stage 4): It was discovered that the Home team was not receiving credit for their goals. This bug was fixed by implementing the appropriate condition to determine whether the attacking team is the Home team.

Possession Did Not Terminate (Stage 4): It was discovered that the possession did not terminate after a goal had been scored. This bug was fixed by implementing the appropriate code to set possessionContinues to False after a goal has been scored.

Testing:

Testing was performed on the soccer dice application in five different test scenarios.

Test 1 - Manual Trace: A manual trace was performed on the Red → Yellow → Green → goal transition, and it was confirmed that the correct rules were followed using the seed value of 23. In addition to verifying that the transition was successful, it was also verified that the score changed appropriately (0,0) → (1,0), and that the possession was switched to the Away team.

Test 2 - Reproducibility: The playmatch(seed=23,numattacks=10) function was run twice, and it was confirmed that the output from both runs were identical, indicating that there were no hidden states in the application.

Test 3 - Edge Cases: Several edge cases were tested, including:

Red "dribble" correctly rolled Red again

Green "corner" correctly looped back to Yellow

Yellow "offside" correctly terminated immediately

Test 4 - Score Accumulation: A manual count was performed of the "GOAL" in a 20-possession match, and it was verified that the final score from the automated application was equal to the manual count.

Test 5 - Robustness: Testing was performed utilizing several different seeds and attacks (1-100), and it was confirmed that the application ran without crashing or entering into an infinite loop.

1. (c) Critical Enhancement with Shot Clock and Performance Analysis

CODE:

```
"""
Soccer Dice: Task 1C Implementation

Simulation of a football/soccer match between two teams with shot clock mechanics.
"""

import random as rand


def roll_die(die_name, die_faces):
    """Roll a die and return the outcome."""
    roll_index = rand.randrange(len(die_faces))
    outcome = die_faces[roll_index]
    print("    Rolled " + die_name + " die: '" + outcome + "'")
    return outcome, roll_index


def play_possession(attacking_team, defending_team, current_score, metrics,
enable_shotclock=False):
    """
    To simulate one complete attacking possession for the attacking_team.
    Tracks the performance metrics and implements the shot clock if it is enabled.
    """

    # Defining the dice faces (unedited)
    red = ['through ball/to yellow', 'inside pass/to blue', 'dribble/throw again',
           'short pass/to black', 'through ball/to green', 'tackled and lost']
    black = ['pass back/to red', 'throw in/to blue', 'shoot',
             'free kick/to yellow', 'long shot at goal', 'tackled and lost']
    blue = ['header at goal', 'shoot', 'opponent shown yellow card',
            'pass back/to red', 'long shot at goal', 'tackled and lost']
    yellow = ['pass back/to black', 'off side', 'tackled and lost',
              'penalty', 'shoot', 'shoot']
    green = ['goal', 'wide', 'goal', 'over bar', 'saved', 'corner/to yellow']

    # Update metrics
    metrics['possessions'] = metrics['possessions'] + 1
    roll_count_in_possession = 0

    # Prints the possession header with current score
    shotclock_status = "ON - " + str(10) + " rolls limit" if enable_shotclock else
"OFF"
    print("\n" + attacking_team + " team attacks (Score: Home " +
          str(current_score[0]) + "-" + str(current_score[1]) + " Away) [Shot Clock:
" + shotclock_status + "]")
    print("  Starting on Red die...")

    # To initialize the possession state
    next_die = 'Red'
    keep_going = True

    while keep_going:

        # TO CHECK SHOT CLOCK
        if enable_shotclock:
            roll_count_in_possession = roll_count_in_possession + 1
            metrics['rolls'] = metrics['rolls'] + 1
```

```
        if roll_count_in_possession > 10:
            print("    -> *** SHOT CLOCK EXPIRED! ***")
            print("    -> Attempting Hail Mary...")

            # Hail Mary: 15% success rate to score
            hail_mary_roll = rand.random()
            if hail_mary_roll < 0.15:
                print("    -> *** HAIL MARY GOAL! ***")
                metrics['goals_scored'] = metrics['goals_scored'] + 1
                metrics['hail_mary_goals'] = metrics['hail_mary_goals'] + 1

                if attacking_team == 'Home':
                    current_score = (current_score[0] + 1, current_score[1])
                else:
                    current_score = (current_score[0], current_score[1] + 1)

                print("    -> Score updated: Home " + str(current_score[0]) +
                      " - " + str(current_score[1]) + " Away")
            else:
                print("    -> Hail Mary missed. Possession lost.")
                metrics['tackles'] = metrics['tackles'] + 1

            keep_going = False
            return current_score, attacking_team  # Other team gets next attack

    # Rolling the appropriate die based on current state
    if next_die == 'Red':
        outcome, idx = roll_die('Red', red)
        if not enable_shotclock:
            metrics['rolls'] = metrics['rolls'] + 1

        if outcome == 'tackled and lost':
            print("    -> " + attacking_team + " loses possession. " +
                  defending_team + " will attack next.")
            metrics['tackles'] = metrics['tackles'] + 1
            keep_going = False
            return current_score, defending_team

        elif outcome == 'through ball/to yellow':
            print("    -> Amazing through ball, switching to Yellow")
            next_die = 'Yellow'

        elif outcome == 'inside pass/to blue':
            print("    -> Neat pass, switching to Blue")
            next_die = 'Blue'

        elif outcome == 'dribble/throw again':
            print("    -> Skilful dribble, rolling Red again")
            next_die = 'Red'

        elif outcome == 'short pass/to black':
            print("    -> Short possession-retaining pass to Black")
            next_die = 'Black'

        elif outcome == 'through ball/to green':
            print("    -> Great pass! Straight to Green for a shot")
            next_die = 'Green'

    elif next_die == 'Black':
        outcome, idx = roll_die('Black', black)
```

```
            if not enable_shotclock:
                metrics['rolls'] = metrics['rolls'] + 1

            if outcome == 'tackled and lost':
                print("    -> Strong tackle! " + defending_team + " regains
possession")
                metrics['tackles'] = metrics['tackles'] + 1
                keep_going = False
                return current_score, defending_team

            elif outcome == 'pass back/to red':
                print("    -> Recycling play, passing back to Red")
                next_die = 'Red'

            elif outcome == 'throw in/to blue':
                print("    -> Throw-in down the line, switching to Blue")
                next_die = 'Blue'

            elif outcome == 'shoot':
                print("    -> Space opens up! Taking a shot — switching to Green")
                metrics['shots_attempted'] = metrics['shots_attempted'] + 1
                next_die = 'Green'

            elif outcome == 'free kick/to yellow':
                print("    -> Foul! Free kick opportunity to Yellow")
                next_die = 'Yellow'

            elif outcome == 'long shot at goal':
                print("    -> Long-shot opportunity! Rolling Green for the attempt")
                metrics['shots_attempted'] = metrics['shots_attempted'] + 1
                next_die = 'Green'

        elif next_die == 'Blue':
            outcome, idx = roll_die('Blue', blue)
            if not enable_shotclock:
                metrics['rolls'] = metrics['rolls'] + 1

            if outcome == 'tackled and lost':
                print("    -> Dispossessed! " + defending_team + " has the ball")
                metrics['tackles'] = metrics['tackles'] + 1
                keep_going = False
                return current_score, defending_team

            elif outcome == 'header at goal':
                print("    -> Head towards goal — to Green")
                metrics['shots_attempted'] = metrics['shots_attempted'] + 1
                next_die = 'Green'

            elif outcome == 'shoot':
                print("    -> Clear sight of goal! Taking a shot — to Green")
                metrics['shots_attempted'] = metrics['shots_attempted'] + 1
                next_die = 'Green'

            elif outcome == 'opponent shown yellow card':
                print("    -> Opponent cautioned! " + attacking_team + " gets a free
shot to Green")
                metrics['shots_attempted'] = metrics['shots_attempted'] + 1
                next_die = 'Green'

            elif outcome == 'pass back/to red':
                print("    -> Recycles the ball back to Red")
```

```
                next_die = 'Red'

            elif outcome == 'long shot at goal':
                print("    -> Attempt from distance towards Green")
                metrics['shots_attempted'] = metrics['shots_attempted'] + 1
                next_die = 'Green'

        elif next_die == 'Yellow':
            outcome, idx = roll_die('Yellow', yellow)
            if not enable_shotclock:
                metrics['rolls'] = metrics['rolls'] + 1

            if outcome == 'off side':
                print("    -> OFFSIDE! Attack ends")
                keep_going = False
                return current_score, defending_team

            elif outcome == 'tackled and lost':
                print("    -> Possession lost in midfield")
                metrics['tackles'] = metrics['tackles'] + 1
                keep_going = False
                return current_score, defending_team

            elif outcome == 'pass back/to black':
                print("    -> Sideways pass to Black")
                next_die = 'Black'

            elif outcome == 'penalty':
                print("    -> Penalty awarded! Direct opportunity to Green")
                metrics['shots_attempted'] = metrics['shots_attempted'] + 1
                next_die = 'Green'

            elif outcome == 'shoot':
                print("    -> Shooting opportunity! Rolling Green")
                metrics['shots_attempted'] = metrics['shots_attempted'] + 1
                next_die = 'Green'

        elif next_die == 'Green':
            outcome, idx = roll_die('Green', green)
            if not enable_shotclock:
                metrics['rolls'] = metrics['rolls'] + 1

            if outcome == 'goal':
                print("    -> *** GOAL! ***")
                metrics['goals_scored'] += 1
                metrics['successful_possessions'] =
metrics['successful_possessions'] + 1

                # To update the score
                if attacking_team == 'Home':
                    current_score = (current_score[0] + 1, current_score[1])
                else:
                    current_score = (current_score[0], current_score[1] + 1)

                print("    -> Score updated: Home " + str(current_score[0]) +
                      " - " + str(current_score[1]) + " Away")
                print("    -> Kick-off to " + defending_team)
                keep_going = False
                return current_score, defending_team

            elif outcome == 'wide':
```

```
                print("    -> Shot goes wide! Goal kick — attack ends")
                keep_going = False
                return current_score, defending_team

            elif outcome == 'over bar':
                print("    -> Over the bar! Goal kick — attack ends")
                keep_going = False
                return current_score, defending_team

            elif outcome == 'saved':
                print("    -> Goalkeeper makes a save! Possession ends")
                keep_going = False
                return current_score, defending_team

            elif outcome == 'corner/to yellow':
                print("    -> Deflection off the goalkeeper — corner awarded!")
                print("    -> Corner kick back to Yellow")
                next_die = 'Yellow'


def play_match(seed, num_attacks, enable_shotclock=False):
    """Orchestrate a complete match between Home and Away teams."""

    # To initialize metrics dictionary
    metrics = {
        'possessions': 0,
        'shots_attempted': 0,
        'goals_scored': 0,
        'tackles': 0,
        'rolls': 0,
        'successful_possessions': 0,
        'hail_mary_goals': 0
    }

    rand.seed(seed)
    score = (0, 0)
    current_attacker = 'Home'
    current_defender = 'Away'

    print("=" * 70)
    print("SOCCER DICE MATCH SIMULATION")
    if enable_shotclock:
        print("SHOT CLOCK ENABLED (10 rolls per possession)")
    else:
        print("SHOT CLOCK DISABLED")
    print("=" * 70)
    print("Kick-off: Home team attacks first")
    print("Simulating " + str(num_attacks) + " attacking possessions")
    print("=" * 70)

    for possession_number in range(1, num_attacks + 1):
        print("\n[Possession " + str(possession_number) + "]")
        score, next_attacker = play_possession(current_attacker, current_defender,
score, metrics, enable_shotclock)
        current_attacker, current_defender = next_attacker, current_attacker

    # Printing final score
    print("\n" + "=" * 70)
    print("FULL-TIME WHISTLE")
    print("=" * 70)
    print("FINAL SCORE: Home " + str(score[0]) + " - " + str(score[1]) + " Away")
```

```
    # Printing performance metrics
    print("\nPERFORMANCE METRICS:")
    print("-" * 70)
    print("Total possessions: " + str(metrics['possessions']))
    print("Total dice rolls: " + str(metrics['rolls']))
    print("Total shots attempted: " + str(metrics['shots_attempted']))
    print("Total goals scored: " + str(metrics['goals_scored']))
    if enable_shotclock:
        print("Hail Mary goals: " + str(metrics['hail_mary_goals']))
    print("Total tackles: " + str(metrics['tackles']))

    if metrics['possessions'] > 0:
        success_rate = 100.0 * metrics['successful_possessions'] /
metrics['possessions']
        print("Possession success rate: {:.2f}%".format(success_rate))

    if metrics['shots_attempted'] > 0:
        shot_accuracy = 100.0 * metrics['goals_scored'] / metrics['shots_attempted']
        print("Shot accuracy: {:.2f}%".format(shot_accuracy))

    if metrics['possessions'] > 0:
        avg_rolls = metrics['rolls'] / metrics['possessions']
        print("Average rolls per possession: {:.2f}".format(avg_rolls))

    print("=" * 70 + "\n")

    return score, metrics


# EXECUTION: Comparision with and without shot clock
print("\n\n")
print("#" * 69)
print("SCENARIO 1: WITHOUT SHOT CLOCK")
print("#" * 69)
score1, metrics1 = play_match(seed=23, num_attacks=5, enable_shotclock=False)

print("\n\n")
print("#" * 69)
print("SCENARIO 2: WITH SHOT CLOCK (10 rolls limit)")
print("#" * 69)
score2, metrics2 = play_match(seed=23, num_attacks=5, enable_shotclock=True)

# COMPARISON
print("\n\n")
print("#" * 69)
print("COMPARATIVE ANALYSIS")
print("#" * 69)
print("Metric                        | Without Clock | With Clock   | Difference")
print("-" * 69)
print("Total Rolls                   | {:<13} | {:<12} | {:<10}".format(
    str(metrics1['rolls']), str(metrics2['rolls']),
    str(metrics2['rolls'] - metrics1['rolls'])
))
print("Goals Scored                  | {:<13} | {:<12} | {:<10}".format(
    str(metrics1['goals_scored']), str(metrics2['goals_scored']),
    str(metrics2['goals_scored'] - metrics1['goals_scored'])
))
print("Shots Attempted               | {:<13} | {:<12} | {:<10}".format(
    str(metrics1['shots_attempted']), str(metrics2['shots_attempted']),
    str(metrics2['shots_attempted'] - metrics1['shots_attempted'])
```

```
    ))
    print("Tackles                    | {:<13} | {:<12} | {:<10}".format(
        str(metrics1['tackles']), str(metrics2['tackles']),
        str(metrics2['tackles'] - metrics1['tackles'])
    ))
    if metrics2['hail_mary_goals'] > 0:
        print("Hail Mary Goals            | {:<13} | {:<12} | {:<10}".format(
            "0", str(metrics2['hail_mary_goals']),
            str(metrics2['hail_mary_goals'])
        ))
    print("=" * 69)
```

OUTPUT:

```
######################################################################
SCENARIO 1: WITHOUT SHOT CLOCK
######################################################################
=====================================================================
SOCCER DICE MATCH SIMULATION
SHOT CLOCK DISABLED
=====================================================================
Kick-off: Home team attacks first
Simulating 5 attacking possessions
=====================================================================

[Possession 1]

Home team attacks (Score: Home 0-0 Away) [Shot Clock: OFF]
  Starting on Red die...
    Rolled Red die: 'dribble/throw again'
    -> Skilful dribble, rolling Red again
    Rolled Red die: 'through ball/to yellow'
    -> Amazing through ball, switching to Yellow
    Rolled Yellow die: 'pass back/to black'
    -> Sideways pass to Black
    Rolled Black die: 'long shot at goal'
    -> Long-shot opportunity! Rolling Green for the attempt
    Rolled Green die: 'goal'
    -> *** GOAL! ***
    -> Score updated: Home 1 - 0 Away
    -> Kick-off to Away

[Possession 2]

Away team attacks (Score: Home 1-0 Away) [Shot Clock: OFF]
  Starting on Red die...
    Rolled Red die: 'short pass/to black'
    -> Short possession-retaining pass to Black
    Rolled Black die: 'free kick/to yellow'
    -> Foul! Free kick opportunity to Yellow
    Rolled Yellow die: 'shoot'
    -> Shooting opportunity! Rolling Green
```

```
    Rolled Green die: 'goal'
    -> *** GOAL! ***
    -> Score updated: Home 1 - 1 Away
    -> Kick-off to Home

[Possession 3]

Home team attacks (Score: Home 1-1 Away) [Shot Clock: OFF]
  Starting on Red die...
    Rolled Red die: 'inside pass/to blue'
    -> Neat pass, switching to Blue
    Rolled Blue die: 'tackled and lost'
    -> Dispossessed! Away has the ball

[Possession 4]

Away team attacks (Score: Home 1-1 Away) [Shot Clock: OFF]
  Starting on Red die...
    Rolled Red die: 'inside pass/to blue'
    -> Neat pass, switching to Blue
    Rolled Blue die: 'opponent shown yellow card'
    -> Opponent cautioned! Away gets a free shot to Green
    Rolled Green die: 'over bar'
    -> Over the bar! Goal kick — attack ends

[Possession 5]

Home team attacks (Score: Home 1-1 Away) [Shot Clock: OFF]
  Starting on Red die...
    Rolled Red die: 'through ball/to yellow'
    -> Amazing through ball, switching to Yellow
    Rolled Yellow die: 'off side'
    -> OFFSIDE! Attack ends

================================================================================
FULL-TIME WHISTLE
================================================================================
FINAL SCORE: Home 1 - 1 Away

PERFORMANCE METRICS:
--------------------------------------------------------------------------------
Total possessions: 5
Total dice rolls: 16
Total shots attempted: 3
Total goals scored: 2
Total tackles: 1
Possession success rate: 40.00%
Shot accuracy: 66.67%
Average rolls per possession: 3.20
================================================================================
```

```
######################################################################
SCENARIO 2: WITH SHOT CLOCK (10 rolls limit)
######################################################################
======================================================================
SOCCER DICE MATCH SIMULATION
SHOT CLOCK ENABLED (10 rolls per possession)
======================================================================
Kick-off: Home team attacks first
Simulating 5 attacking possessions
======================================================================

[Possession 1]

Home team attacks (Score: Home 0-0 Away) [Shot Clock: ON - 10 rolls limit]
  Starting on Red die...
    Rolled Red die: 'dribble/throw again'
    -> Skilful dribble, rolling Red again
    Rolled Red die: 'through ball/to yellow'
    -> Amazing through ball, switching to Yellow
    Rolled Yellow die: 'pass back/to black'
    -> Sideways pass to Black
    Rolled Black die: 'long shot at goal'
    -> Long-shot opportunity! Rolling Green for the attempt
    Rolled Green die: 'goal'
    -> *** GOAL! ***
    -> Score updated: Home 1 - 0 Away
    -> Kick-off to Away

[Possession 2]

Away team attacks (Score: Home 1-0 Away) [Shot Clock: ON - 10 rolls limit]
  Starting on Red die...
    Rolled Red die: 'short pass/to black'
    -> Short possession-retaining pass to Black
    Rolled Black die: 'free kick/to yellow'
    -> Foul! Free kick opportunity to Yellow
    Rolled Yellow die: 'shoot'
    -> Shooting opportunity! Rolling Green
    Rolled Green die: 'goal'
    -> *** GOAL! ***
    -> Score updated: Home 1 - 1 Away
    -> Kick-off to Home

[Possession 3]

Home team attacks (Score: Home 1-1 Away) [Shot Clock: ON - 10 rolls limit]
  Starting on Red die...
    Rolled Red die: 'inside pass/to blue'
    -> Neat pass, switching to Blue
    Rolled Blue die: 'tackled and lost'
```

-> Dispossessed! Away has the ball

[Possession 4]

Away team attacks (Score: Home 1-1 Away) [Shot Clock: ON - 10 rolls limit]
Starting on Red die...
Rolled Red die: 'inside pass/to blue'
-> Neat pass, switching to Blue
Rolled Blue die: 'opponent shown yellow card'
-> Opponent cautioned! Away gets a free shot to Green
Rolled Green die: 'over bar'
-> Over the bar! Goal kick — attack ends

[Possession 5]

Home team attacks (Score: Home 1-1 Away) [Shot Clock: ON - 10 rolls limit]
Starting on Red die...
Rolled Red die: 'through ball/to yellow'
-> Amazing through ball, switching to Yellow
Rolled Yellow die: 'off side'
-> OFFSIDE! Attack ends

==========
FULL-TIME WHISTLE
==========
FINAL SCORE: Home 1 - 1 Away

PERFORMANCE METRICS:

Total possessions: 5
Total dice rolls: 16
Total shots attempted: 3
Total goals scored: 2
Hail Mary goals: 0
Total tackles: 1
Possession success rate: 40.00%
Shot accuracy: 66.67%
Average rolls per possession: 3.20
==========
##########
COMPARATIVE ANALYSIS
##########
Metric | Without Clock | With Clock | Difference

Total Rolls | 16 | 16 | 0
Goals Scored | 2 | 2 | 0
Shots Attempted | 3 | 3 | 0
Tackles | 1 | 1 | 0
==========

Conclusion Based on Actual Results

The comparison of the soccer dice simulation with a shot clock and one with no shot clock based upon the code and outputs presented here demonstrates that in this specific instance (using the seed number =23, and for the 5 possessions) the inclusion of a shot clock did not appear to have an effect on the primary metrics when using this data set. However, this lack of a measurable impact does provide insight as to the systems' design and/or stability.

Detailed Comparative Analysis

Metric	Without Shot Clock	With Shot Clock	Difference	Impact
Total Rolls	16	16	0	No change
Goals Scored	2	2	0	No change
Shots Attempted	3	3	0	No change
Tackles	1	1	0	No change
Possession Success Rate	40.00%	40.00%	0%	No change
Shot Accuracy	66.67%	66.67%	0%	No change
Average Rolls/Possession	3.20	3.20	0	No change
Hail Mary Goals	N/A	0	N/A	Not triggered

Task 2. Critically assess, select and apply data science libraries and algorithms (MLO3)

2. (a) Problem Definition and Manual Analysis

Problem Statement:

Data Science Use Case: Perform Sentiment Analysis on Unstructured Text Data to Classify Tone of Voice in First Year Students' Written Reviews.

University Specific Problem Context:

Coventry University receives written reviews from its first-year students regarding its facilities including library, cafeteria, accommodation etc. These reviews are currently held within a Microsoft Excel spreadsheet and have never been systematically reviewed. The University could develop a system which will allow the analysis of student opinion in order to identify specific issues and positive aspects of its services.

Primary Question: Is it possible to automatically classify a number of reviews into Positive, Neutral and Negative categories in order to determine whether student satisfaction is improving over time?

Sample Fresher Context

Rao is a Coventry University fresher who is studying Computer Science. In his first year at university Rao wrote six reviews of the university library after visiting it on numerous occasions:

The Six Reviews

Review 1 (Week 2):

"The library is absolutely amazing! The staff are incredibly helpful, the Wi-Fi is fast, and there are plenty of quiet study areas. I'm so grateful to have this resource on campus."

Review 2 (Week 4):

"Terrible experience today. Too crowded, couldn't find a seat anywhere, and the computers are constantly freezing. Very disappointing and frustrating."

Review 3 (Week 6):

"The library has the books I need and the computers usually work. It's adequate for my studies but nothing exceptional. Could use better facilities."

Review 4 (Week 8):

"Brilliant new study pods! Perfect for group work and very quiet. The extended opening hours during exam season are fantastic. Really impressed with the improvements."

Review 5 (Week 10):

"Very disappointed with noise levels during peak hours. People talking loudly, not enough enforcement of quiet zones. The atmosphere is too chaotic for serious studying."

Review 6 (Week 12):

"Excellent resource! Best place on campus for concentrated work. The librarians are supportive and knowledgeable. Highly recommend for any student needing a productive study environment."

Manual “By Hand” Solution:

I would manually apply the same logical steps to classify the six reviews without a computer:

Step 1: Establish Criteria for Classifying Reviews

I would define criteria to help guide my classification decisions prior to reviewing each of the six reviews:

POSITIVE: Contains complimentary statements, indicates the reviewer is pleased, expresses admiration for the staff or facilities.

NEGATIVE: Indicates dissatisfaction, expresses frustration/disappointment, identifies problems/issues.

NEUTRAL: Provides balanced comments, acknowledges both positives and negatives, expresses neither positive nor negative comments.

Step 2: Determine Emotionally Charged Words Within Each Review

Review 1 Analysis:

Identified words expressing a strong positive emotion:

Words: “absolutely amazing,” “incredibly helpful,” “grateful,” “fast,” “plenty”

Identified words indicating a negative emotion:

No words were identified.

Sentiment Tone: Extremely positive

Classification: POSITIVE

Review 2 Analysis:

Identified words expressing a negative emotion:

Words: “Terrible,” “Couldn’t find”, “Constantly freezing”, “Disappointing,” “Frustrating”

Identified words expressing a positive emotion:

No words were identified.

Sentiment Tone: Extremely negative

Classification: NEGATIVE

Review 3 Analysis:

Identified words expressing a positive emotion:

Words: “Has the books I need,” “Adequate.”

Identified words expressing a negative emotion:

Words: “Nothing exceptional,” “Could use better.”

Sentiment Tone: Overall mixed sentiment, slightly negative

Classification: NEUTRAL

Review 4 Analysis:

Identified words expressing a positive emotion:

Words: “Brilliant”, “Perfect”, “Fantastic,” “Impressed,” “Really Impressed”.

Identified words expressing a negative emotion:

No words were identified.

Sentiment Tone: Extremely positive

Classification: POSITIVE

Review 5 Analysis:

Identified words expressing a positive emotion:

No words were identified.

Identified words expressing a negative emotion:

Words: “Very Disappointed”, “Not enough”, “Too Chaotic”.

Sentiment Tone: Negative

Classification: NEGATIVE

Review 6 Analysis:

Identified words expressing a positive emotion:

Words: “Excellent,” “Best,” “Supportive,” “Knowledgeable,” “Highly Recommend”.

Identified words expressing a negative emotion:

No words were identified.

Sentiment Tone: Extremely positive

Classification: POSITIVE

Step 3: Calculate Summary Statistics

Summary Statistics of Overall Sentiment:

Positive Reviews: 3 / 6 = 50%

Negative Reviews: 2 / 6 = 33%

Neutral Reviews: 1 / 6 = 17%

Emerging Trends:

Generally speaking Rao expressed a positive view towards the university library.

Major concerns were expressed in Week 4 and Week 10 (Crowded and Noise Issues).

Positive feedback increased after the introduction of new facilities (Study Pods and Extended Hours).

Step 4: Derive Actionable Insights

Actionable Insights for Institutional Decision Makers:

Strength: The library has provided good support through its staff and new facilities.

Problem: There is a major issue with overcrowding during peak hours.

Trend: The introduction of new library facilities and support systems (Study Pods, Extended Hours) has had a positive impact on the student body.

Reason Why This is a Data Science Problem

Unstructured Data: The reviews contain free-form text and therefore lack a consistent or structured format.

Recognising Patterns: Recognising that words such as “Amazing,” “Terrible,” “Adequate” etc., relate to a particular type of sentiment pattern.

Classification: Categorise each review as either Positive, Neutral, or Negative based upon the content of each review.

Challenge of Scalability: Manually classifying 6 reviews is feasible. However, if the university wanted to classify 600 or even 6000 reviews from all students attending the university, manual classification would become impossible.

Practical Impact: The results of this project will provide the university with the information needed to make informed decisions regarding the distribution of resources and the improvement of its facilities.

2. **(b) Automated Implementation Using NLTK and Pandas**

NLTK vs Pandas: A Brief Comparison

NLTK (Natural Language Toolkit) & Pandas are two different libraries for text analysis that complement each other.

NLTK Abilities:

NLTK is a toolkit for Natural Language Processing (NLP) primarily designed to process large amounts of text and includes several important abilities including:

Tokenization – Text is broken down into single “words” so that each individual item can be analysed separately.

Stop Word Removal – Common words (e.g., “the”, “and”) are removed from the text so that the analysis is focused only on content related terms.

Stemming – Words are reduced to their base form to help prevent over counting identical concepts.

NLTK also comes with several built-in NLP tools and datasets which allows users to easily perform complex tasks such as text preprocessing without having to build those tools themselves. As mentioned above NLTK's focus is strictly on text and does not have any statistical or data aggregation functionality.

Pandas Abilities:

Pandas is a powerful tool for manipulating and analysing data. Pandas is able to organize data into table structures called Data Frames where rows represent observations and columns represent the variables measured for each observation. Pandas is also capable of performing a wide variety of calculations such as means, counts, percentiles etc. Additionally, Pandas is well suited for working with both numerical and categorical data. Pandas also works very well with structured data; however, Pandas is not suitable for much in the way of text processing.

Key Differences:

Aspect	**NLTK**	**Pandas**
Focus	Text processing	Data analysis
Tokenization	Yes	No
Statistical Analysis	Limited	Extensive

Aspect	NLTK	Pandas
DataFrames	No	Yes
Stop Word Removal	Yes	No

Alternative Library Combinations:

Alternative	Pros	Cons
TextBlob + Pandas	TextBlob has built-in sentiment	Less control over algorithm
spaCy + Pandas	Faster than NLTK	More complex
NLTK + NumPy	NumPy good for calculations	No table structure like DataFrame
Manual + Pandas	Full control over preprocessing	Reinventing the wheel (tokenization is complex)

Basic Setup for NLTK and Pandas:

1. **NLTK Library**

The code begins with the following lines which import all of the needed libraries.

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
```

Explanation:

import nltk: This line imports all of the needed NLTK libraries for downloading the required data resources.

wordtokenize: This function will be used to break down a string of characters into each individual character (or in this case word).

stopwords: This is a built-in list of the most commonly used words in English that are often used when filtering out common words such as "the" "a" "an" etc.

2. Importing The Libraries - Pandas

I also used the pandas library in my next section and used a standard alias for it, imported as "pd"

```
import pandas as pd
```

Explanation:

import pandas as pd: This line imports the pandas library with the standard alias "pd".

This allowed me to create data frames to organize and analyse my results.

3. Downloading The Required NLTK Data Resources

Once I imported the required libraries, I had to download the required data files from the NLTK library before they can be used for the first time.

```
nltk.download('punkt')        # Tokenization models
nltk.download('punkt_tab')    # Updated tokenization data
nltk.download('stopwords')    # Common English stop words
```

What Each of These Do:

punkt: This contains pre-trained models for breaking down strings of text into individual words and sentences.

punkttab: This is an updated model of punkt which was necessary because of how the newer versions of NLTK operate.

stopwords: This is a list of the top 179 most commonly used words in English that I used for the filtering process in the next section.

CODE:

```
"""
Task 2B: Automated Sentiment Analysis Using NLTK and Pandas

"""

import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import pandas as pd

# Definition of sentiment words
positive_words = [
    ('amazing', 2), ('excellent', 2), ('brilliant', 2), ('fantastic', 2),
    ('helpful', 2), ('grateful', 1), ('impressed', 1), ('best', 2),
    ('love', 2), ('perfect', 2), ('wonderful', 2), ('great', 1),
    ('good', 1), ('recommend', 2), ('fast', 1), ('supportive', 2),
    ('knowledgeable', 1), ('productive', 1), ('adequate', 1)
]

negative_words = [
    ('terrible', 2), ('disappointing', 2), ('disappointed', 2),
```

```
    ('frustrated', 1), ('frustrating', 2), ('crowded', 2),
    ('freezing', 1), ('chaotic', 2), ('inadequate', 1),
    ('poor', 1), ('bad', 1), ('awful', 2), ('horrible', 2),
    ('useless', 2), ('noise', 1), ('problem', 1)
]

# Rao's six reviews stored as a list
reviews = [
    "The library is absolutely amazing! The staff are incredibly helpful, the Wi-Fi is fast, and there are plenty of quiet study areas. I'm so grateful to have this resource on campus.",

    "Terrible experience today. Too crowded, couldn't find a seat anywhere, and the computers are constantly freezing. Very disappointing and frustrating.",

    "The library has the books I need and the computers usually work. It's adequate for my studies but nothing exceptional. Could use better facilities.",

    "Brilliant new study pods! Perfect for group work and very quiet. The extended opening hours during exam season are fantastic. Really impressed with the improvements.",

    "Very disappointed with noise levels during peak hours. People talking loudly, not enough enforcement of quiet zones. The atmosphere is too chaotic for serious studying.",

    "Excellent resource! Best place on campus for concentrated work. The librarians are supportive and knowledgeable. Highly recommend for any student needing a productive study environment."
]


def preprocess_text_nltk(text):
    """
    Using NLTK to preprocess text: tokenize, lowercase, remove stop words.

    """
    # Tokenizing using NLTK
    tokens = word_tokenize(text)

    # To get English stop words from NLTK
    stop_words = set(stopwords.words('english'))

    # Filtering tokens: lowercase and to remove stop words
    filtered_tokens = []
    for word in tokens:
        word_lower = word.lower()
        if word_lower not in stop_words and word_lower.isalpha():
            filtered_tokens.append(word_lower)

    return filtered_tokens


def classify_sentiment_weighted(review_text):
    """
    Classification of sentiment using weighted word scoring with NLTK preprocessing.

    """
    # Preprocessing text using NLTK
    tokens = preprocess_text_nltk(review_text)
```

```
    # Calculation of weighted positive score
    pos_score = 0
    pos_words_found = []
    for word, weight in positive_words:
        count = tokens.count(word)
        if count > 0:
            pos_score = pos_score + count * weight
            pos_words_found.append(word)

    # Calculation of weighted negative score
    neg_score = 0
    neg_words_found = []
    for word, weight in negative_words:
        count = tokens.count(word)
        if count > 0:
            neg_score = neg_score + count * weight
            neg_words_found.append(word)

    # Calculation of net sentiment score
    net_score = pos_score - neg_score

    # Classification based on thresholds
    if net_score > 2:
        sentiment = 'POSITIVE'
    elif net_score < -2:
        sentiment = 'NEGATIVE'
    else:
        sentiment = 'NEUTRAL'

    return {
        'sentiment': sentiment,
        'pos_score': pos_score,
        'neg_score': neg_score,
        'net_score': net_score,
        'pos_words': len(pos_words_found),
        'neg_words': len(neg_words_found)
    }


def analyze_reviews_with_pandas(reviews):
    """
    To analyze reviews and store results in a Pandas DataFrame.

    """
    # Creating empty lists to store results
    review_numbers = []
    review_texts = []
    sentiments = []
    pos_scores = []
    neg_scores = []
    net_scores = []
    pos_word_counts = []
    neg_word_counts = []

    # Classification of each review
    for i in range(len(reviews)):
        review = reviews[i]
        result = classify_sentiment_weighted(review)

        review_numbers.append(i + 1)
        review_texts.append(review[:50] + "...")  # First 50 chars
```

```
        sentiments.append(result['sentiment'])
        pos_scores.append(result['pos_score'])
        neg_scores.append(result['neg_score'])
        net_scores.append(result['net_score'])
        pos_word_counts.append(result['pos_words'])
        neg_word_counts.append(result['neg_words'])

    # Creating Pandas DataFrame
    df = pd.DataFrame({
        'Review': review_numbers,
        'Text': review_texts,
        'Sentiment': sentiments,
        'Pos_Score': pos_scores,
        'Neg_Score': neg_scores,
        'Net_Score': net_scores,
        'Pos_Words': pos_word_counts,
        'Neg_Words': neg_word_counts
    })

    return df


# MAIN EXECUTION
print("=" * 79)
print("SENTIMENT ANALYSIS USING NLTK AND PANDAS")
print("=" * 79)

# Analysis of reviews and storing them in DataFrame
results_df = analyze_reviews_with_pandas(reviews)

# Displaying the DataFrame
print("\nReview Analysis Results:")
print("-" * 79)
print(results_df.to_string(index=False))

# Summary of statistics using Pandas
print("\n" + "=" * 79)
print("SUMMARY STATISTICS (Using Pandas)")
print("=" * 79)

sentiment_counts = results_df['Sentiment'].value_counts()
print("\nSentiment Distribution:")
for sentiment in ['POSITIVE', 'NEGATIVE', 'NEUTRAL']:
    if sentiment in sentiment_counts:
        count = sentiment_counts[sentiment]
        percentage = 100.0 * count / len(results_df)
        print("{}: {} reviews ({:.1f}%)".format(sentiment, count, percentage))
    else:
        print("{}: 0 reviews (0.0%)".format(sentiment))

# Additional statistics using Pandas
print("\nAverage Scores:")
print("Average Positive Score: {:.2f}".format(results_df['Pos_Score'].mean()))
print("Average Negative Score: {:.2f}".format(results_df['Neg_Score'].mean()))
print("Average Net Score: {:.2f}".format(results_df['Net_Score'].mean()))

print("\n" + "=" * 79)
```

Output:

```
================================================================================
SENTIMENT ANALYSIS USING NLTK AND PANDAS
================================================================================

Review Analysis Results:
--------------------------------------------------------------------------------
 Review                                Text Sentiment  Pos_Score  Neg_Score  Net_Score  Pos_Words
Neg_Words
     1 The library is absolutely amazing! The staff are i...  POSITIVE          6          0          6          4          0
     2 Terrible experience today. Too crowded, couldn't f...  NEGATIVE          0          9         -9          0
5
     3 The library has the books I need and the computers...   NEUTRAL          1          0          1          1
0
     4 Brilliant new study pods! Perfect for group work a...  POSITIVE          7          0          7          4          0
     5 Very disappointed with noise levels during peak ho...  NEGATIVE          0          5         -5          0
3
     6 Excellent resource! Best place on campus for conce...  POSITIVE         10          0         10          6
0

================================================================================
SUMMARY STATISTICS (Using Pandas)
================================================================================

Sentiment Distribution:
POSITIVE: 3 reviews (50.0%)
NEGATIVE: 2 reviews (33.3%)
NEUTRAL: 1 reviews (16.7%)

Average Scores:
Average Positive Score: 4.00
Average Negative Score: 2.33
Average Net Score: 1.67

================================================================================
```

2. (c) Critical Evaluation of Libraries

Introduction

Student sentiment analysis was addressed through the use of two libraries: NLTK for the purpose of Natural Language Processing and Pandas for the purpose of data manipulation. As both libraries play critical roles in forming the core of the analysis pipeline, as well as having distinct strengths and limitations, they each serve different roles within the overall analysis pipeline.

Evaluating NLTK

NLTK performs exceptionally well at the pre-processing of text. It tokenizes raw reviews, eliminates stop words and prepares the reviews for analysis. Through these pre-processing activities, the subsequent sentiment scoring function can then solely focus upon the meaningful words. NLTK has been widely adopted throughout academia and research and includes numerous pre-built resources such as stop word lists, which greatly reduces the need for manually implementing them. Unfortunately, there is no built-in sentiment classifier available in the syllabus using NLTK, so the user will need to define their own logic to determine positive or negative sentiment based upon the results, which limits NLTK's out-of-the box capabilities for performing exactly this task.

Evaluating Pandas

Pandas offers several benefits when it comes to efficiently storing review texts, scores, and classifications, including DataFrames. In addition to being able to store this information quickly and efficiently, Pandas also offers rapid summary, aggregation and reporting of results. For example, functions such as .valuecounts(), .mean() etc., provide users with the ability to obtain immediate insight into the number of reviews that are classified as positive, negative, or neutral. One of the primary advantages of Pandas is its flexibility, supporting projects ranging from a handful of reviews to thousands of reviews. Although Pandas offers excellent performance and scalability, it has very limited features for analysing text and therefore requires additional tools to perform any level of language analysis and/or pre-process text.

Suitability

Together, NLTK and Pandas, represent a comprehensive and practical pipeline that meets the requirements of the syllabus:

• NLTK addresses the unstructured text phase of the project.

• Pandas serves to manage structured, reproducible analytics and reporting.

Although neither library would be capable of providing an effective solution individually, together they enable the automated, transparent and scalable analysis of sentiment in student reviews, making them an appropriate combination for academic and institutional applications.

Therefore, based upon the student review problem analysed here, the combined application of both libraries is justifiable and highly recommended.

References:

References
Downey, A. B. (2024). Think Python: How to Think Like a Computer Scientist. O'Reilly Media.

Coventry University. (2025). 7042SCN Programming for Data Science - Module Summary. Coventry University.

Coventry University. (2025). 7042SCN Lab 01A: Python Lab Worksheet - Values, Types, Functions. Coventry University.

Coventry University. (2025). 7042SCN Lab 01B: Python Lab Worksheet - Data Types, Control Flow, Lists, and Loops. Coventry University.

Coventry University. (2025). 7042SCN Lab 02A: Python Lab Worksheet - Data Structures and Loops. Coventry University.

Coventry University. (2025). 7042SCN Lab 02B: Python Lab Worksheet - Dice Game Probability and String Search Algorithms. Coventry University.

Coventry University. (2025). 7042SCN Lab 03A: Numpy and Matplotlib Lab Worksheet. Coventry University.

Coventry University. (2025). 7042SCN Lab 03B: Python Lab Worksheet - Computational Thinking and Dice Game. Coventry University.

Coventry University. (2025). 7042SCN Lab 04A: Introduction to Pandas and Exploratory Data Analysis. Coventry University.

Coventry University. (2025). 7042SCN Lab 05B: Introduction to Natural Language Processing with NLTK. Coventry University.

Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. O'Reilly Media.

McKinney, W. (2022). Python for Data Analysis (3rd ed.). O'Reilly Media.

VanderPlas, J. (2016). Python Data Science Handbook: Essential Tools for Working with Data. O'Reilly Media.

Bates, S. (2020). A Whirlwind Tour of Python. O'Reilly Media. Available at: https://jakevdp.github.io/WhirlwindTourOfPython/

Python Software Foundation. (2025). Python 3 Documentation. Retrieved from https://docs.python.org/3/

NLTK Project. (2025). Natural Language Toolkit Documentation. Retrieved from https://www.nltk.org/

Pandas Development Team. (2025). Pandas Documentation. Retrieved from https://pandas.pydata.org/