7143CEM PROGRAMMING FOR DATA SCIENCE

Individual Portfolio

Student Name: Sujan Tumbaraguddi
Student ID: 14194733

Contents

Task 1. Explain, critique, comment, and debug code (MLO2)

(a) The given code lacks ':' for 'def', 'while', 'for', and 'if' statements, resulting in an instant error. The indentation in the given code is of 2 spaces, but it should be of 4 spaces. Because there is no code that calls the function 'whatdoido(L)', the function is loaded but never performed. And from now on, anytime the function 'whatdoido(L)' is called for execution, the flag 's' is set to 'True', and it is set to 'False' once it enters the 'while' loop. Keeping in mind that 'while' loop stops only when the flag 's' is 'False', the first 'for' loop has a condition that check if the element L(i) is less than or equal to the element L(i+1) where 'i' varies from 1 and goes up to, but does not include, len(L)-3 (with step of +1), if the condition is True then the flag 's' is set to 'True'. The second 'for' loop checks the same condition and manipulates the flag 's', but this time 'i' varies from len(L)-3 and goes up to but does not include 1 (with a step of -1).
So, whenever the elements in a list (or) tuple (or) etc., in range 1 to len(L)-3 are in descending order, the 'while' loop comes to end or else the function will execute till the end of time (provided we supply the electricity and computing power;). Also, the function doesn't return anything, it only manipulates the flag 's' provided the sequence feed has elements more than or equal to 5.

Final code (ignoring range in both 'for' loops):

```
#########################################################################
#INDIVIDUAL TASK 1a. Explain, critique, comment, and debug code (MLO2)
#########################################################################

def whatdoido(L):
    '''Arrange values in Descending order for index 1 to len(L)-2'''
    s = True                                    # s is a flag.
    while (s):                                  # while loop runs till the end of sort.
        s = False
        for i in range(1,len(L)-3):             # sort the sequence range from 1 to len(L)-3.
            if (L[i] < L[i+1]):
                # now swap values L[i] and L[i+1]
                L[i], L[i+1] = L[i+1], L[i]
                s = True

        for i in range(len(L)-3,1,-1):          # sort the sequence range from len(L)-3 to 1.
            if (L[i] < L[i+1]):
                # now swap values L[i] and L[i+1]
                L[i], L[i+1] = L[i+1], L[i]
                s = True
    return(L)                                   # returns the sorted sequence.
```

(b) Firstly,

```
#########################################################################
```

```
#INDIVIDUAL TASK 1b. Explain, critique, comment, and debug code (MLO2)
#########################################################################

def whatdoido(L):
    '''Arrange values in Descending order for index 1 to len(L)-2'''
    s = True                                    # s is a flag.
    print('A flag is used to stop the "while" loop that is used to sort the whole
sequence and it is set to', s, 'to initiate the while loop.')
    comp = 0
    swap = 0
    print('There are 2 counters to count no. of comparisons and swaps that happened in
the code.')
    while (s):                                  # while loop runs till the end of sort.
        s = False
        print('Flag is set to',s,'after entering "while" loop.')
        for i in range(1,len(L)-3):             # sort the sequence range from 1 to len(L)-3.
            comp = comp + 1
            print('comparison Count =',comp)
            print('Elements in comparison are',L[i],'and',L[i+1],'.')
            if (L[i] < L[i+1]):
                # now swap values L[i] and L[i+1]
                print('As',L[i],'is less than',L[i+1],', we are swapping them.')
                L[i], L[i+1] = L[i+1], L[i]
                swap = swap + 1
                print('Swap count =',swap)
                s = True
                print('Flag is set to',s,'after swap.')

        for i in range(len(L)-3,1,-1):          # sort the sequence range from len(L)-3 to 1.
            comp = comp + 1
            print('comparison Count =',comp)
            print('Elements in comparison are',L[i],'and',L[i+1],'.')
            if (L[i] < L[i+1]):
                # now swap values L[i] and L[i+1]
                print('As',L[i],'is less than',L[i+1],', we are swapping them.')
                L[i], L[i+1] = L[i+1], L[i]
                swap = swap + 1
                print('Swap count =',swap)
                s = True
                print('Flag is set to',s,'after swap.')
        print('Sequence at the end of a "while" loop iteration is',L,'.')
    return(L)                                   # returns the sorted sequence.


L = [10,9,1,7,6,5,4,9,6,0] #['g','f','a','d','c','b','a']

print('To Arrange values in Descending order for index 1 to len(L)-2 in the given
sequence.')
print('The sequence that was sent to sort is', L,'.')

print('After sorting, the sequence is', whatdoido(L),'.')
```

Output:

To Arrange values in Descending order for index 1 to len(L)-2 in the given sequence.

The sequence that was sent to sort is [10, 9, 1, 7, 6, 5, 4, 9, 6, 0] .
A flag is used to stop the "while" loop that is used to sort the whole sequence and it is set to True to initiate the while loop.
There are 2 counters to count no. of comparisons and swaps that happened in the code.
Flag is set to False after entering "while" loop.
comparison Count = 1
Elements in comparison are 9 and 1 .
comparison Count = 2
Elements in comparison are 1 and 7 .
As 1 is less than 7 , we are swapping them.
Swap count = 1
Flag is set to True after swap.
comparison Count = 3
Elements in comparison are 1 and 6 .
As 1 is less than 6 , we are swapping them.
Swap count = 2
Flag is set to True after swap.
comparison Count = 4
Elements in comparison are 1 and 5 .
As 1 is less than 5 , we are swapping them.
Swap count = 3
Flag is set to True after swap.
comparison Count = 5
Elements in comparison are 1 and 4 .
As 1 is less than 4 , we are swapping them.
Swap count = 4
Flag is set to True after swap.
comparison Count = 6
Elements in comparison are 1 and 9 .
As 1 is less than 9 , we are swapping them.
Swap count = 5
Flag is set to True after swap.
comparison Count = 7
Elements in comparison are 1 and 6 .
As 1 is less than 6 , we are swapping them.
Swap count = 6
Flag is set to True after swap.
comparison Count = 8
Elements in comparison are 9 and 6 .
comparison Count = 9
Elements in comparison are 4 and 9 .
As 4 is less than 9 , we are swapping them.
Swap count = 7
Flag is set to True after swap.
comparison Count = 10
Elements in comparison are 5 and 9 .
As 5 is less than 9 , we are swapping them.
Swap count = 8
Flag is set to True after swap.

comparison Count = 11
Elements in comparison are 6 and 9 .
As 6 is less than 9 , we are swapping them.
Swap count = 9
Flag is set to True after swap.
comparison Count = 12
Elements in comparison are 7 and 9 .
As 7 is less than 9 , we are swapping them.
Swap count = 10
Flag is set to True after swap.
Sequence at the end of a "while" loop iteration is [10, 9, 9, 7, 6, 5, 4, 6, 1, 0] .
Flag is set to False after entering "while" loop.
comparison Count = 13
Elements in comparison are 9 and 9 .
comparison Count = 14
Elements in comparison are 9 and 7 .
comparison Count = 15
Elements in comparison are 7 and 6 .
comparison Count = 16
Elements in comparison are 6 and 5 .
comparison Count = 17
Elements in comparison are 5 and 4 .
comparison Count = 18
Elements in comparison are 4 and 6 .
As 4 is less than 6 , we are swapping them.
Swap count = 11
Flag is set to True after swap.
comparison Count = 19
Elements in comparison are 4 and 1 .
comparison Count = 20
Elements in comparison are 6 and 4 .
comparison Count = 21
Elements in comparison are 5 and 6 .
As 5 is less than 6 , we are swapping them.
Swap count = 12
Flag is set to True after swap.
comparison Count = 22
Elements in comparison are 6 and 6 .
comparison Count = 23
Elements in comparison are 7 and 6 .
comparison Count = 24
Elements in comparison are 9 and 7 .
Sequence at the end of a "while" loop iteration is [10, 9, 9, 7, 6, 6, 5, 4, 1, 0] .
Flag is set to False after entering "while" loop.
comparison Count = 25
Elements in comparison are 9 and 9 .
comparison Count = 26
Elements in comparison are 9 and 7 .
comparison Count = 27

Elements in comparison are 7 and 6 .
comparison Count = 28
Elements in comparison are 6 and 6 .
comparison Count = 29
Elements in comparison are 6 and 5 .
comparison Count = 30
Elements in comparison are 5 and 4 .
comparison Count = 31
Elements in comparison are 4 and 1 .
comparison Count = 32
Elements in comparison are 5 and 4 .
comparison Count = 33
Elements in comparison are 6 and 5 .
comparison Count = 34
Elements in comparison are 6 and 6 .
comparison Count = 35
Elements in comparison are 7 and 6 .
comparison Count = 36
Elements in comparison are 9 and 7 .
Sequence at the end of a "while" loop iteration is [10, 9, 9, 7, 6, 6, 5, 4, 1, 0] .
After sorting, the sequence is [10, 9, 9, 7, 6, 6, 5, 4, 1, 0] .

Secondly,

```
#########################################################################
#INDIVIDUAL TASK 1b. Explain, critique, comment, and debug code (MLO2)
#########################################################################

def whatdoido(L):
    '''Arrange values in Descending order for index 1 to len(L)-3'''
    s = True                                    # s is a flag.
    print('A flag is used to stop the "while" loop that is used to sort the whole
sequence and it is set to', s, 'to initiate the while loop.')
    comp = 0
    swap = 0
    print('There are 2 counters to count no. of comparisons and swaps that happened in
the code.')
    while (s):                                  # while loop runs till the end of sort.
        s = False
        print('Flag is set to',s,'after entering "while" loop.')
        for i in range(1,len(L)-3):             # sort the sequence range from 1 to len(L)-3.
            comp = comp + 1
            print('comparison Count =',comp)
            print('Elements in comparison are',L[i],'and',L[i+1],'.')
            if (L[i] < L[i+1]):
                # now swap values L[i] and L[i+1]
                print('As',L[i],'is less than',L[i+1],', we are swapping them.')
                L[i], L[i+1] = L[i+1], L[i]
                swap = swap + 1
                print('Swap count =',swap)
                s = True
                print('Flag is set to',s,'after swap.')
```

```
        print('Sequence at the end of a "while" loop iteration is',L,'.')
    return(L)                                   # returns the sorted sequence.


L = [10,9,1,7,6,5,4,9,6,0] #['g','f','a','d','c','b','a']

print('To Arrange values in Descending order for index 1 to len(L)-3 in the given
sequence.')
print('The sequence that was sent to sort is', L,'.')

print('After sorting, the sequence is', whatdoido(L),'.')
```

Output:

To Arrange values in Descending order for index 1 to len(L)-3 in the given sequence.
The sequence that was sent to sort is [10, 9, 1, 7, 6, 5, 4, 9, 6, 0] .
A flag is used to stop the "while" loop that is used to sort the whole sequence and it is set to True to initiate the while loop.
There are 2 counters to count no. of comparisons and swaps that happened in the code.
Flag is set to False after entering "while" loop.
comparison Count = 1
Elements in comparison are 9 and 1 .
comparison Count = 2
Elements in comparison are 1 and 7 .
As 1 is less than 7 , we are swapping them.
Swap count = 1
Flag is set to True after swap.
comparison Count = 3
Elements in comparison are 1 and 6 .
As 1 is less than 6 , we are swapping them.
Swap count = 2
Flag is set to True after swap.
comparison Count = 4
Elements in comparison are 1 and 5 .
As 1 is less than 5 , we are swapping them.
Swap count = 3
Flag is set to True after swap.
comparison Count = 5
Elements in comparison are 1 and 4 .
As 1 is less than 4 , we are swapping them.
Swap count = 4
Flag is set to True after swap.
comparison Count = 6
Elements in comparison are 1 and 9 .
As 1 is less than 9 , we are swapping them.
Swap count = 5
Flag is set to True after swap.
Sequence at the end of a "while" loop iteration is [10, 9, 7, 6, 5, 4, 9, 1, 6, 0] .
Flag is set to False after entering "while" loop.
comparison Count = 7
Elements in comparison are 9 and 7 .

comparison Count = 8
Elements in comparison are 7 and 6 .
comparison Count = 9
Elements in comparison are 6 and 5 .
comparison Count = 10
Elements in comparison are 5 and 4 .
comparison Count = 11
Elements in comparison are 4 and 9 .
As 4 is less than 9 , we are swapping them.
Swap count = 6
Flag is set to True after swap.
comparison Count = 12
Elements in comparison are 4 and 1 .
Sequence at the end of a "while" loop iteration is [10, 9, 7, 6, 5, 9, 4, 1, 6, 0] .
Flag is set to False after entering "while" loop.
comparison Count = 13
Elements in comparison are 9 and 7 .
comparison Count = 14
Elements in comparison are 7 and 6 .
comparison Count = 15
Elements in comparison are 6 and 5 .
comparison Count = 16
Elements in comparison are 5 and 9 .
As 5 is less than 9 , we are swapping them.
Swap count = 7
Flag is set to True after swap.
comparison Count = 17
Elements in comparison are 5 and 4 .
comparison Count = 18
Elements in comparison are 4 and 1 .
Sequence at the end of a "while" loop iteration is [10, 9, 7, 6, 9, 5, 4, 1, 6, 0] .
Flag is set to False after entering "while" loop.
comparison Count = 19
Elements in comparison are 9 and 7 .
comparison Count = 20
Elements in comparison are 7 and 6 .
comparison Count = 21
Elements in comparison are 6 and 9 .
As 6 is less than 9 , we are swapping them.
Swap count = 8
Flag is set to True after swap.
comparison Count = 22
Elements in comparison are 6 and 5 .
comparison Count = 23
Elements in comparison are 5 and 4 .
comparison Count = 24
Elements in comparison are 4 and 1 .
Sequence at the end of a "while" loop iteration is [10, 9, 7, 9, 6, 5, 4, 1, 6, 0] .
Flag is set to False after entering "while" loop.

comparison Count = 25
Elements in comparison are 9 and 7 .
comparison Count = 26
Elements in comparison are 7 and 9 .
As 7 is less than 9 , we are swapping them.
Swap count = 9
Flag is set to True after swap.
comparison Count = 27
Elements in comparison are 7 and 6 .
comparison Count = 28
Elements in comparison are 6 and 5 .
comparison Count = 29
Elements in comparison are 5 and 4 .
comparison Count = 30
Elements in comparison are 4 and 1 .
Sequence at the end of a "while" loop iteration is [10, 9, 9, 7, 6, 5, 4, 1, 6, 0] .
Flag is set to False after entering "while" loop.
comparison Count = 31
Elements in comparison are 9 and 9 .
comparison Count = 32
Elements in comparison are 9 and 7 .
comparison Count = 33
Elements in comparison are 7 and 6 .
comparison Count = 34
Elements in comparison are 6 and 5 .
comparison Count = 35
Elements in comparison are 5 and 4 .
comparison Count = 36
Elements in comparison are 4 and 1 .
Sequence at the end of a "while" loop iteration is [10, 9, 9, 7, 6, 5, 4, 1, 6, 0] .
After sorting, the sequence is [10, 9, 9, 7, 6, 5, 4, 1, 6, 0] .

After deleting the second 'for' loop from the code, we get the "Code with example" and "output" shown above. We can see from this:

- The number of comparisons remains the same, i.e., 36 in output with and without the second 'for' loop. (Comparison count will differ with different example for codes with and without the second 'for' loop)
- The swap count is reduced from 12 to 9.
- In both programmes, whether with and without the second 'for' loop, the first element is not considered for comparison or swapping.
- In the code with two 'for' loops, only the last element is not considered for comparison and swapping, but in the code with one 'for' loop, the last two elements are not considered for comparison and swapping.
- The sequence at the end is different.

Conclusion: When both 'for' loops are used, elements are sorted in descending order, with the first and last elements excluded. We sort elements in descending order when we just utilise the first 'for' loop, excluding the first and last two elements. Because the number of elements considered for sorting is reduced in code with only the first 'for' loop, the number of comparisons and swapping is reduced compared to code with two 'for' loops. The sequencing at the end will also be different.

Task 2. Design, build and test (MLO2)

(a):

#1:

Planning:

- Checking the required packages to import, which is "random" as this is required for rolling both the dice and getting an outcome from the 6 predefined values on each dice.
- To create global variables like "batting" and "umpire" as a list which contain ['1','2','3','4','6','owzthat'] and ['bowled','stumped','caught','lbw','no ball','not out'] respectively.
- Define a function "cricket", which handles all the tasks like rolling dice, conditions to handle score, no ball, wicket(out).

Requirements: One player & 2 dice-based game to be played for 10 balls with 2 wickets. Roll 2 dice named 'batting' dice and 'umpire' dice. Before starting the match, the number of balls to be bowled and the number of wickets available are set (so balls and wickets are the resources to be used up over the simulated cricket match). Each time a ball is bowled, the player rolls the batting dice. If the batting dice lands on 1, 2, 3, 4 or 6, that number of runs is added to the total runs scored. However, if the batting dice lands on "owzthat" then the player rolls the umpire dice to determine the outcome of the appeal. If the umpire dice lands on "bowled", "stumped", "caught" or "lbw" then the batter is out (the player loses one wicket) and no runs are scored. The match finishes when either all the balls have been bowled or all the wickets have been lost (whichever comes first).

Analysis & implementation: Import the "random" package. Initiate Global variables, "batting" and "umpire" as a list which contain ['1','2','3','4','6','owzthat'] and ['bowled','stumped','caught','lbw','no ball','not out'], "ball" and "wicket" as '10' and '2'. Define a function named "cricket", and under that function define a 'for' loop to ball all the 10 balls. Define a condition to check if the total number of wickets is reached i.e., 2. If yes, then break the 'for' loop irrespective of the number of balls bowled. If no, continue until all 10 balls are bowled. While we continue to play balls, we should roll the 'batting' dice first using 'random.choice()'. Using 'if – elif' condition and based on the outcome, if it's one of ['1','2','3','4','6'], then add it to the total score and continue to next ball. If the outcome is ['owzthat'], then roll the 'umpire' diced using 'random.choice()'. Using 'if – elif' condition and based on the outcome, if it's one of ['bowled','stumped','caught','lbw'], then a wicket is lost.

#2:

Planning:

- Initialize a local variable inside the function 'cricket' called 'score' and set it to 0(int).
- Define a separate function called 'batting' for rolling a 'batting' dice and 'umpire' dice as it is used frequently until all balls are bowled or all wickets are lost. Here, we pass 'score' and 'wicket' variable back and forth from 'cricket' function so that score and wicket are updated when every ball is bowled.
- Renaming the global variables "batting" and "umpire" to "batting_options" and "umpire_options", if not this will conflict with the function 'batting'.

Requirements: N/A

Analysis & implementation: Initialize a local variable inside the function 'cricket' called 'score' and set it to 0(int). Define a function named "batting" and pass 'score' and 'wicket value from 'cricket' function. Define a condition to check if the total number of wickets is reached i.e., 2. If yes, then break the 'for' loop irrespective of the number of balls bowled. If no, continue until all 10 balls are bowled. While we continue to play balls, we should roll the 'batting' dice first using 'random.choice()'. Using 'if – elif' condition and based on the outcome, if it's one of ['1','2','3','4','6'], then add it to the total score and continue to next ball. If the outcome is ['owzthat'], then roll the 'umpire' diced using 'random.choice()'. Using 'if – elif' condition and based on the outcome, if it's one of ['bowled','stumped','caught','lbw'], then a wicket is lost. And then pass back the updated 'score' and 'wicket' back to the 'cricket' function. Rename the global variables "batting" and "umpire" to "batting_options" and "umpire_options".

#3:
Planning:

- If the umpire dice lands on "not out" then no wickets are lost, and no runs are scored, can be implemented in an additional 'elif' condition in the 'batting' function.
- If the umpire dice lands on "no ball" then the player adds one run to the total runs scored and the ball must be bowled again. To handle an additional ball, define a function named 'no_ball' and call the 'batting' function to ball an additional ball.

Requirements: N/A
Analysis & implementation: If it's ['not out'], then continue with the next ball using 'elif' under the current if-elif conditions present after we get 'umpire_options'.
Define a "no_ball" function with input variables (score, wicket) and returns updated variables as (score, wicket) and inside this add 1 run to the score variable, and the call "batting" function which balls an additional ball and updates the score with the run scored for that additional ball.

#4:
Planning:

- Update Python code should include enough print statements to generate a clear nontrivial example illustrating a play of the match, telling the story as the dice is rolled and the match score changes.

Requirements: N/A
Analysis & implementation: Adding print statements and using the global and local variables inside the print statements wherever necessary. Also, adding print statements which would give runs scored, number of balls bowled, umpire decisions, total score (runs / wickets) for every ball, and so on.

```
######################################################################
#INDIVIDUAL TASK 2a. Explain, critique, comment, and debug code (MLO2)
######################################################################

import random

'''Dice-based cricket match simulation Owzthat for one player
   One dice (called the "batting" dice) is labelled 1, 2, 3, 4, "owzthat" and 6.
   A second dice (called the "umpire" dice) is labelled "bowled", "stumped",
   "caught", "not out", "no ball" and "lbw". Before starting the match, the
   number of balls to be bowled and the number of wickets available are set
   (so balls and wickets are the resources to be used up over the simulated
    cricket match). Each time a ball is bowled, the player rolls the batting
   dice. If the batting dice lands on 1, 2, 3, 4 or 6, that number of runs is
   added to the total runs scored. However, if the batting dice lands on
   "owzthat" then the player rolls the umpire dice to determine the outcome
   of the appeal. If the umpire dice lands on "bowled", "stumped", "caught"
   or "lbw" then the batter is out (the player loses one wicket) and no runs
   are scored. If the umpire dice lands on "not out" then no wickets are lost
   and no runs are scored. If the umpire dice lands on "no ball" then the
   player adds one run to the total runs scored and the ball must be bowled
   again. The match finishes when either all the balls have been bowled or
   all the wickets have been lost (whichever comes first). The aim is to
   maximize the total number of runs scored'''

print('Dice-based cricket match simulation Owzthat for one player.\n')

batting_options = ['1','2','3','4','6','owzthat']
umpire_options = ['bowled','stumped','caught','lbw','no ball','not out']
```

```
def cricket(ball, wickets):
    print('Number of balls and wickets set for the match are', ball, 'and', wickets,
'respectively.\n')

    print('Note:')
    print("1. The batting dice contains the following values:
['1','2','3','4','6','owzthat']. and is rolled first for each ball until the
predetermined number of balls (or) wickets is reached.")
    print("2. The Umpire dice has ['bowled','stumped','caught','lbw','no ball','not out']
and is rolled when the batting dice reveals 'owzthat'.\n")
    score = 0
    wicket = 0
    print('\nScore is set to (Runs/Wickets): ( 0 / 0 )\n')
    for i in range (1,ball + 1):
        if wicket == wickets:
            break
        else:
            score, wicket = batting(score, wicket)
            print('Score : (', score, '/', wicket, ')')
            print('Number of balls bowled:',i)
    print('\nTotal score:', score, '/', wicket)

def batting(score, wicket):
    run = random.choice(batting_options)
    print("\nNow rolling the batting dice.")
    if run == '1' or run == '2' or run == '3' or run == '4' or run == '6':
        score += int(run)
        print('Runs scored :', int(run))
    elif run == 'owzthat':
        print('Chance of a wicket as we got an appeal "owzthat" from the batting dice.')
        decision = random.choice(umpire_options)
        print("Umpire Dice will be rolled now.")
        if decision == 'bowled' or decision == 'stumped' or decision =='caught' or
decision == 'lbw':
            wicket += 1
            print("And it's a wicket. Incredibly", decision, ".")
        elif decision == 'no ball':
            print("Seems like a 'No Ball' here. An additional ball will be bowled and 1
run is added to the score.")
            score, wicket = no_ball(score, wicket)
        elif decision == 'not out':
            print('Umpire dice says "Not Out".')
    return score, wicket

def no_ball(score, wicket):
    score += 1
    score, wicket = batting(score, wicket)
    return score, wicket

cricket(10, 2)
```

OUTPUT:

Sample #1:

Dice-based cricket match simulation Owzthat for one player.

Number of balls and wickets set for the match are 10 and 2 respectively.

Note:
1. The batting dice contains the following values: ['1','2','3','4','6','owzthat']. and is rolled first for each ball until the predetermined number of balls (or) wickets is reached.
2. The Umpire dice has ['bowled','stumped','caught','lbw','no ball','not out'] and is rolled when the batting dice reveals 'owzthat'.

Score is set to (Runs/Wickets): (0 / 0)

Now rolling the batting dice.
Chance of a wicket as we got an appeal "owzthat" from the batting dice.
Umpire Dice will be rolled now.
And it's a wicket. Incredibly lbw .
Score : (0 / 1)
Number of balls bowled: 1

Now rolling the batting dice.
Runs scored : 6
Score : (6 / 1)
Number of balls bowled: 2

Now rolling the batting dice.
Runs scored : 1
Score : (7 / 1)
Number of balls bowled: 3

Now rolling the batting dice.
Runs scored : 4
Score : (11 / 1)
Number of balls bowled: 4

Now rolling the batting dice.
Runs scored : 2
Score : (13 / 1)
Number of balls bowled: 5

Now rolling the batting dice.
Runs scored : 4
Score : (17 / 1)
Number of balls bowled: 6

Now rolling the batting dice.
Runs scored : 4
Score : (21 / 1)
Number of balls bowled: 7

Now rolling the batting dice.
Runs scored : 4
Score : (25 / 1)
Number of balls bowled: 8

Now rolling the batting dice.
Runs scored : 3
Score : (28 / 1)
Number of balls bowled: 9

Now rolling the batting dice.
Chance of a wicket as we got an appeal "owzthat" from the batting dice.
Umpire Dice will be rolled now.
And it's a wicket. Incredibly bowled .
Score : (28 / 2)
Number of balls bowled: 10

Total score: 28 / 2

Sample #2:

Dice-based cricket match simulation Owzthat for one player.

Number of balls and wickets set for the match are 10 and 2 respectively.

Note:
1. The batting dice contains the following values: ['1','2','3','4','6','owzthat']. and is rolled first for each ball until the predetermined number of balls (or) wickets is reached.
2. The Umpire dice has ['bowled','stumped','caught','lbw','no ball','not out'] and is rolled when the batting dice reveals 'owzthat'.

Score is set to (Runs/Wickets): (0 / 0)

Now rolling the batting dice.
Runs scored : 4
Score : (4 / 0)
Number of balls bowled: 1

Now rolling the batting dice.
Runs scored : 4
Score : (8 / 0)
Number of balls bowled: 2

Now rolling the batting dice.
Runs scored : 1
Score : (9 / 0)
Number of balls bowled: 3

Now rolling the batting dice.
Runs scored : 4
Score : (13 / 0)
Number of balls bowled: 4

Now rolling the batting dice.
Runs scored : 6
Score : (19 / 0)
Number of balls bowled: 5

Now rolling the batting dice.
Runs scored : 4
Score : (23 / 0)
Number of balls bowled: 6

Now rolling the batting dice.
Chance of a wicket as we got an appeal "owzthat" from the batting dice.
Umpire Dice will be rolled now.
Umpire dice says "Not Out".
Score : (23 / 0)
Number of balls bowled: 7

Now rolling the batting dice.
Runs scored : 2
Score : (25 / 0)
Number of balls bowled: 8

Now rolling the batting dice.
Runs scored : 2
Score : (27 / 0)
Number of balls bowled: 9

Now rolling the batting dice.
Runs scored : 2
Score : (29 / 0)
Number of balls bowled: 10

Total score: 29 / 0

(b):

```
######################################################################
#INDIVIDUAL TASK 2a. Explain, critique, comment, and debug code (MLO2)
######################################################################

import random

'''Dice-based cricket match simulation Owzthat for one player
```

```
    One dice (called the "batting" dice) is labelled 1, 2, 3, 4, "owzthat" and 6.
    A second dice (called the "umpire" dice) is labelled "bowled", "stumped",
    "caught", "not out", "no ball" and "lbw". Before starting the match, the
    number of balls to be bowled and the number of wickets available are set
    (so balls and wickets are the resources to be used up over the simulated
     cricket match). Each time a ball is bowled, the player rolls the batting
    dice. If the batting dice lands on 1, 2, 3, 4 or 6, that number of runs is
    added to the total runs scored. However, if the batting dice lands on
    "owzthat" then the player rolls the umpire dice to determine the outcome
    of the appeal. If the umpire dice lands on "bowled", "stumped", "caught"
    or "lbw" then the batter is out (the player loses one wicket) and no runs
    are scored. If the umpire dice lands on "not out" then no wickets are lost
    and no runs are scored. If the umpire dice lands on "no ball" then the
    player adds one run to the total runs scored and the ball must be bowled
    again. The match finishes when either all the balls have been bowled or
    all the wickets have been lost (whichever comes first). The aim is to
    maximize the total number of runs scored'''

print('Dice-based cricket match simulation Owzthat for one player.\n')

batting_options = ['1','2','3','4','6','owzthat']
umpire_options = ['bowled','stumped','caught','lbw','no ball','not out']

def cricket(ball, wickets):
    print('Number of balls and wickets set for the match are', ball, 'and', wickets,
'respectively.\n')

    print('Note:')
    print("1. The batting dice contains the following values:
['1','2','3','4','6','owzthat']. and is rolled first for each ball until the
predetermined number of balls (or) wickets is reached.")
    print("2. The Umpire dice has ['bowled','stumped','caught','lbw','no ball','not out']
and is rolled when the batting dice reveals 'owzthat'.\n")
    print("3. The Variant game has an additional dice umpire as ['no ball','not out','run
out'].\n")
    for j in range(0,2):
        score, wicket, num_no_ball, runs_no_ball, not_out_decisions = 0, 0, 0, 0, 0
        if j == 0:
            print('\nORIGINAL OWZTHAT GAME')
        else:
            print('\nVARIANT GAME')
        print('\nScore is set to (Runs/Wickets): ( 0 / 0 )\n')
        for i in range (1,ball + 1):
            if wicket == wickets:
                overs_played = [(i-1) // 6, (i-1) % 6]
                break
            else:
                score, wicket, j, num_no_ball, runs_no_ball, not_out_decisions =
batting(score, wicket, j, num_no_ball, runs_no_ball, not_out_decisions)
                print('Score : (', score, '/', wicket, ')')
                print('Number of balls bowled:',i)
                overs_played = [ball // 6, ball % 6]
        print('\nTotal score:', score, '/', wicket)
        print('Total no-balls bowled:', num_no_ball)
        print('Total runs scored in no-balls:', runs_no_ball)
        print('Total overs played:', overs_played[0], '.', overs_played[1], 'overs')
```

```
        print("Total number of 'not out' decisions made by umpire dice:",
not_out_decisions)

def batting(score, wicket, j, num_no_ball, runs_no_ball, not_out_decisions):
    run = random.choice(batting_options)
    print("\nNow rolling the batting dice.")
    if run == '1' or run == '2' or run == '3' or run == '4' or run == '6':
        score += int(run)
        print('Runs scored :', int(run))
    elif run == 'owzthat':
        print('Chance of a wicket as we got an appeal "owzthat" from the batting dice.')
        decision = random.choice(umpire_options)
        print("Umpire Dice will be rolled now.")
        if decision == 'bowled' or decision == 'stumped' or decision =='caught' or
decision == 'lbw':
            wicket += 1
            print("And it's a wicket. Incredibly", decision, ".")
        elif decision == 'no ball':
            if j == 0:
                num_no_ball += 1
                score, wicket, j, num_no_ball, runs_no_ball, not_out_decisions =
no_ball(score, wicket, j, num_no_ball, runs_no_ball, not_out_decisions)
            elif  j == 1:
                score, wicket, j, num_no_ball, runs_no_ball, not_out_decisions =
real_no_ball(score, wicket, j, num_no_ball, runs_no_ball, not_out_decisions)
        elif decision == 'not out':
            print('Umpire dice says "Not Out".')
            not_out_decisions += 1
    return score, wicket, j, num_no_ball, runs_no_ball, not_out_decisions

def no_ball(score, wicket, j, num_no_ball, runs_no_ball, not_out_decisions):
    print("Seems like a 'No Ball' here. An additional ball will be bowled and 1 run is
added to the score.")
    score += 1
    score1 = score
    print('Score : (', score, '/', wicket, ')')
    score, wicket, j, num_no_ball, runs_no_ball, not_out_decisions = batting(score,
wicket, j, num_no_ball, runs_no_ball, not_out_decisions)
    runs_no_ball = score - score1 + 1
    return score, wicket, j, num_no_ball, runs_no_ball, not_out_decisions

def real_no_ball(score, wicket, j, num_no_ball, runs_no_ball, not_out_decisions):
    num_no_ball += 1
    runs_no_ball += 1
    print("Seems like a 'No Ball' here. An additional ball will be bowled and 1 run is
added to the score.")
    print("And it's a 'Free Hit'.")
    score += 1
    print('Score : (', score, '/', wicket, ')')
    real_umpire_options = ['no ball','not out','run out']
    run = random.choice(batting_options)
    print("Now rolling the batting dice.")
    if run == '1' or run == '2' or run == '3' or run == '4' or run == '6':
        score += int(run)
        runs_no_ball += int(run)
        print("Runs scored in 'Free Hit' :", int(run))
```

```
    elif run == 'owzthat':
        print('Chance of a wicket as we got an appeal "owzthat" from the batting dice.')
        real_decision = random.choice(real_umpire_options)
        print("Umpire Dice will be rolled now, which is specialized for additional
ball.")
        if real_decision == 'run out':
            wicket += 1
            print("And it's a wicket. Incredibly", real_decision, ".")
        elif real_decision == 'no ball':
            run = 'owzthat'
            while run == 'owzthat':
                run = random.choice(batting_options)
            score += int(run)
            runs_no_ball += int(run)
            print('Runs scored in the additional ball :', int(run))
            score, wicket, j, num_no_ball, runs_no_ball, not_out_decisions =
real_no_ball(score, wicket, j, num_no_ball, runs_no_ball, not_out_decisions)
        elif real_decision == 'not out':
            print('Umpire dice says "Not Out".')
            not_out_decisions += 1
            run = 'owzthat'
            while run == 'owzthat':
                run = random.choice(batting_options)
            score += int(run)
            runs_no_ball += int(run)
            print("Runs scored in 'Free Hit' :", int(run))
    return score, wicket, j, num_no_ball, runs_no_ball, not_out_decisions

cricket(10, 2)
```

OUTPUT:

Dice-based cricket match simulation Owzthat for one player.

Number of balls and wickets set for the match are 10 and 2 respectively.

Note:
1. The batting dice contains the following values: ['1','2','3','4','6','owzthat']. and is rolled first for each ball until the predetermined number of balls (or) wickets is reached.
2. The Umpire dice has ['bowled','stumped','caught','lbw','no ball','not out'] and is rolled when the batting dice reveals 'owzthat'.
3. The Variant game has an additional dice umpire as ['no ball','not out','run out'].

ORIGINAL OWZTHAT GAME

Score is set to (Runs/Wickets): (0 / 0)

Now rolling the batting dice.
Runs scored : 4
Score : (4 / 0)

Number of balls bowled: 1

Now rolling the batting dice.
Runs scored : 3
Score : (7 / 0)
Number of balls bowled: 2

Now rolling the batting dice.
Chance of a wicket as we got an appeal "owzthat" from the batting dice.
Umpire Dice will be rolled now.
Seems like a 'No Ball' here. An additional ball will be bowled and 1 run is added to the score.
Score : (8 / 0)
……..

Total score: 27 / 0
Total no-balls bowled: 1
Total runs scored in no-balls: 2
Total overs played: 1 . 4 overs
Total number of 'not out' decisions made by umpire dice: 0

VARIANT GAME

Score is set to (Runs/Wickets): (0 / 0)

Now rolling the batting dice.
Chance of a wicket as we got an appeal "owzthat" from the batting dice.
Umpire Dice will be rolled now.
Seems like a 'No Ball' here. An additional ball will be bowled and 1 run is added to the score.
And it's a 'Free Hit'.
Score : (1 / 0)
Now rolling the batting dice.
Runs scored in 'Free Hit' : 3
Score : (4 / 0)
Number of balls bowled: 1
…….

Total score: 32 / 0
Total no-balls bowled: 1
Total runs scored in no-balls: 4
Total overs played: 1 . 4 overs
Total number of 'not out' decisions made by umpire dice: 0

Summary Table:

Counts/Game	Original	Variant
Total score	27 / 0	32 / 0
Total no-balls bowled	1 ball	1 ball
Total runs scored in no-balls	2 runs	4 runs

Total overs played	1.4 overs	1.4 overs
Total number of 'not out' decisions made by umpire dice	0	0

Conclusion:

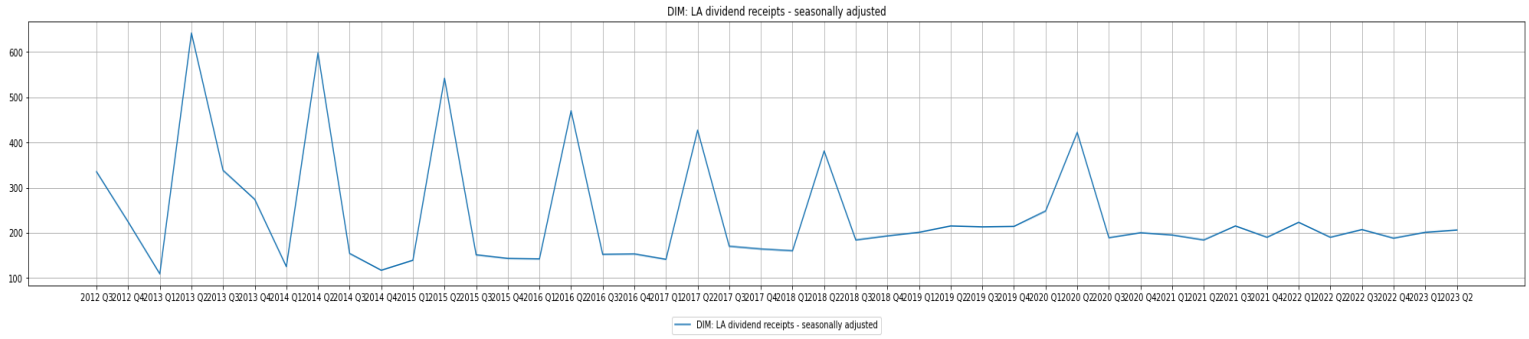
Even though the total number of balls played in the 'Original' and 'Variant' games is the same, this just means that the 'umpire' dice did not land on the possibilities where a wicket would be lost, and the wickets lost in the Total score is also Zero. The overall number of runs scored varies as 27 and 32. We must also consider the runs scored during 'no-ball' in this case. Because there is no 'Free Hit' option in the additional ball played, the player must play wisely. In the variant game, a 'Free Hit' is awarded for each additional ball bowled, so this ball has the potential to dramatically change the match if the player takes full use of it. According to the following summary, both games consist of 1 no ball, but the runs scored during this are 2 and 4, respectively, indicating that the player was nervous during the additional ball, which may have affected the balls played after the additional ball in the original game as well. Considering the Total Score, Variant Game has the highest scores.

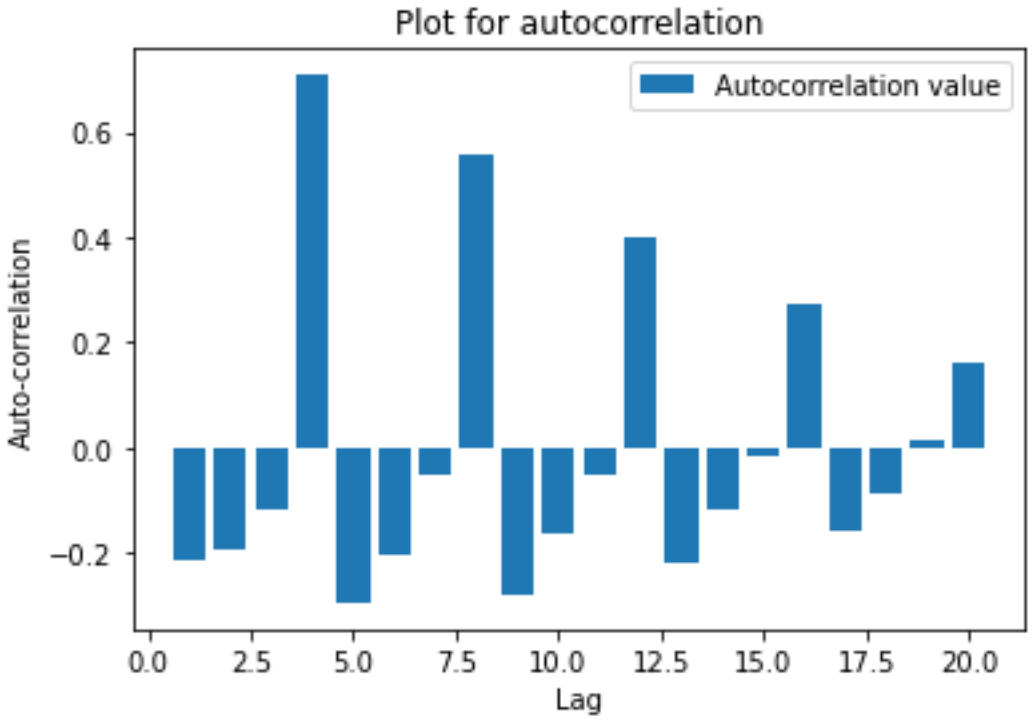
Task 3. Critically assess, select, and apply data science tools (MLO3)

(a):

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load time series data from a CSV file into a pandas DataFrame
time_series = 'LA_dividend_receipts.csv'  # File containing LA Dividend Receipts data
(DIM: LA Dividend Receipts - Seasonally Adjusted - Office for National Statistics, 2023)
data = pd.read_csv(time_series)
time_series_data = data.to_numpy()  # Convert the DataFrame to a NumPy array
print(time_series_data)

# Extract a specific variable from the time series data
variable = time_series_data[:, 1]

# Plot the time series data
plt.figure(figsize=(35, 5))
plt.plot(time_series_data[:, 0], time_series_data[:, 1], label='AWE: Construction Level
(£): Seasonally Adjusted Bonus')
plt.grid()
plt.title('DIM: LA dividend receipts - seasonally adjusted')
plt.legend(loc='lower center', bbox_to_anchor=(0.5, -0.2))
plt.show()

# Define a function to calculate autocorrelation for a given variable and lag
def autocorrelation(variable, lag):              # OpenAI's GPT-3 model
    n = len(variable)
    mean = np.mean(variable)
    num = np.sum((variable[:n - lag] - mean) * (variable[lag:] - mean))
    deno = np.sum((variable - mean) ** 2)
    return num / deno
```

```
# Calculate autocorrelation values for different lags
lags = range(1, 21)
autocorrelations = [autocorrelation(variable, lag) for lag in lags]

# Plot the autocorrelation values
plt.bar(lags, autocorrelations, label='Autocorrelation value')
plt.legend()
plt.xlabel('Lag')
plt.ylabel('Auto-correlation')
plt.title('Plot for autocorrelation')
plt.show()
```





Summary of Autocorrelation Plot:

Positive Autocorrelation:
Positive correlations between the quarterly dividend receipts and their historical values are indicated by peaks in the autocorrelation plot.
In particular, a significant positive autocorrelation at a specific lag suggests that the dividend receipts will likely exhibit a similar trend in upcoming quarters.

Potential Seasonality:
Peaks that appear on a regular basis indicate that the data may contain seasonality or recurrent patterns.
Determining these trends might be important in predicting when dividend payments will increase or decrease.

Decision-Making Implications:
Knowing autocorrelation makes it easier to predict dividend payments in the future using past trends.
It aids in making strategic decisions by offering information about the dataset's level of predictability or persistence.

Areas for Further Investigation:
Peaks at lags might call for more research on outside variables influencing dividend receipts, such the state of the economy or changes in policy.

Conclusion:
An effective method for identifying temporal relationships in LA dividend payouts is the autocorrelation plot.
By applying these results, we hope to strengthen our decision-making procedures and improve our capacity to predict variations in dividend payments.

(b):

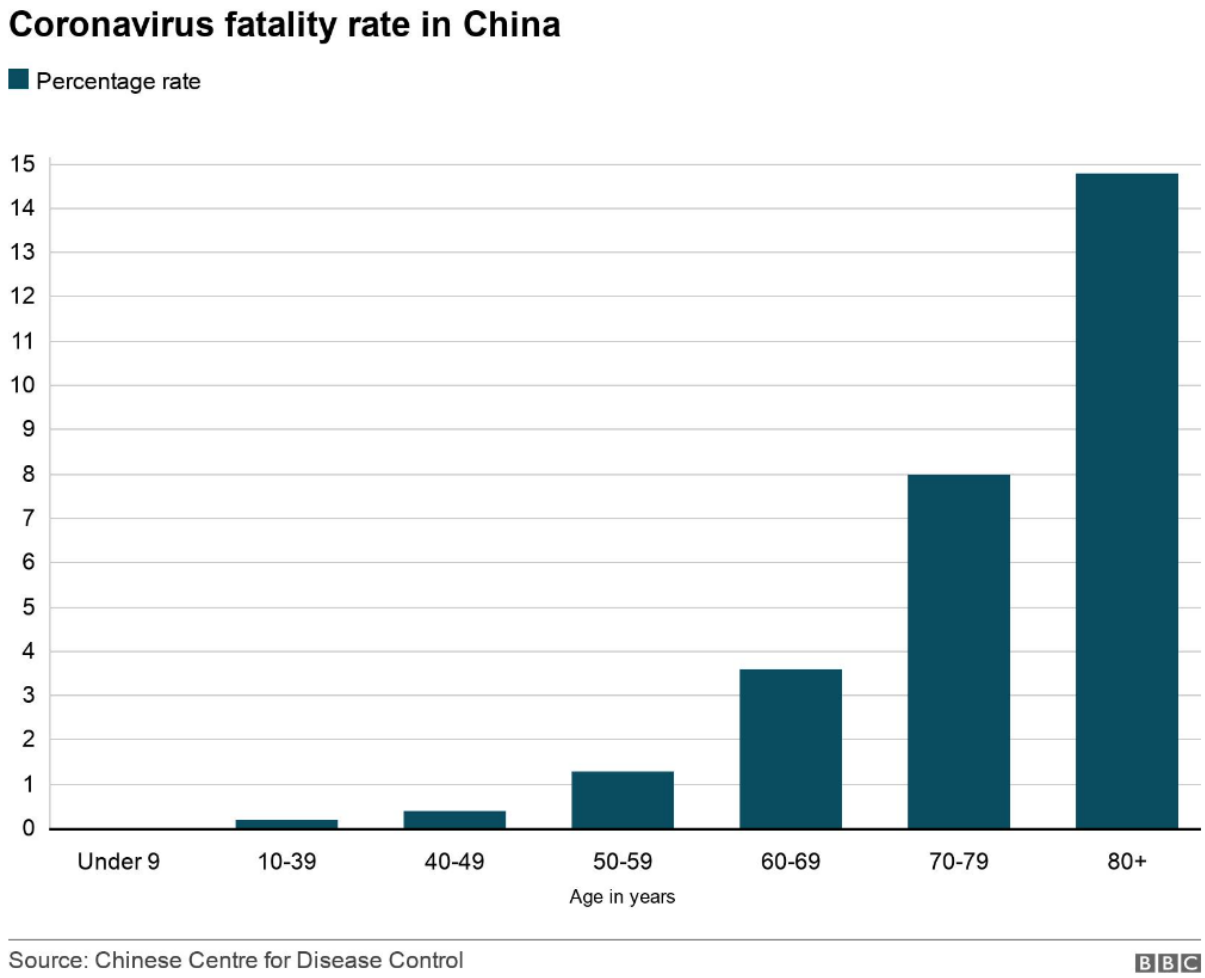


Image: Coronavirus fatality rate in China (BBC News, 2020)

Baseline characteristics	Confirmed cases, N (%)	Deaths, N (%)	Case fatality rate, %	Observed time, PD	Mortality, per 10 PD
Overall Age, years	44,672	1,023	2.3	6,61,609	0.015
0–9	416 (0.9)	–	–	4,383	–
10–19	549 (1.2)	1 (0.1)	0.2	6,625	0.002
20–29	3,619 (8.1)	7 (0.7)	0.2	53,953	0.001
30–39	7,600 (17.0)	18 (1.8)	0.2	1,14,550	0.002
40–49	8,571 (19.2)	38 (3.7)	0.4	1,28,448	0.003
50–59	10,008 (22.4)	130 (12.7)	1.3	1,51,059	0.009
60–69	8,583 (19.2)	309 (30.2)	3.6	1,28,088	0.024
70–79	3,918 (8.8)	312 (30.5)	8	55,832	0.056
≥80	1,408 (3.2)	208 (20.3)	14.8	18,671	0.111

TABLE 1. Patients, deaths, and case fatality rates, as well as observed time and mortality for n=44,672 confirmed COVID-19 cases in Mainland China as of February 11, 2020. (Weekly, 2020)

Using Seaborn and Matplotlib:

```
# Importing relevant libraries
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Creating a DataFrame from the provided data
cases = {
    'Age': ['0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-69', '70-79', '≥80'],
    'Fatality': [0, 0.2, 0.2, 0.2, 0.4, 1.3, 3.6, 8, 14.8]
}

dataset = pd.DataFrame(cases)

# Setting the style of seaborn
sns.set(style="whitegrid")

# Creating the bar plot using Seaborn
plt.figure(figsize=(10, 6))  # Defining figure size
bar_plot = sns.barplot(x='Age', y = 'Fatality', data=dataset, palette='viridis')

# Rotating x-axis labels for better readability
```

```
bar_plot.set_xticklabels(bar_plot.get_xticklabels())

# Adding labels and title
plt.xlabel('Agein years')
plt.title('Coronavirus fatality rate in China')

# Displaying the plot
plt.tight_layout()
plt.show()
```

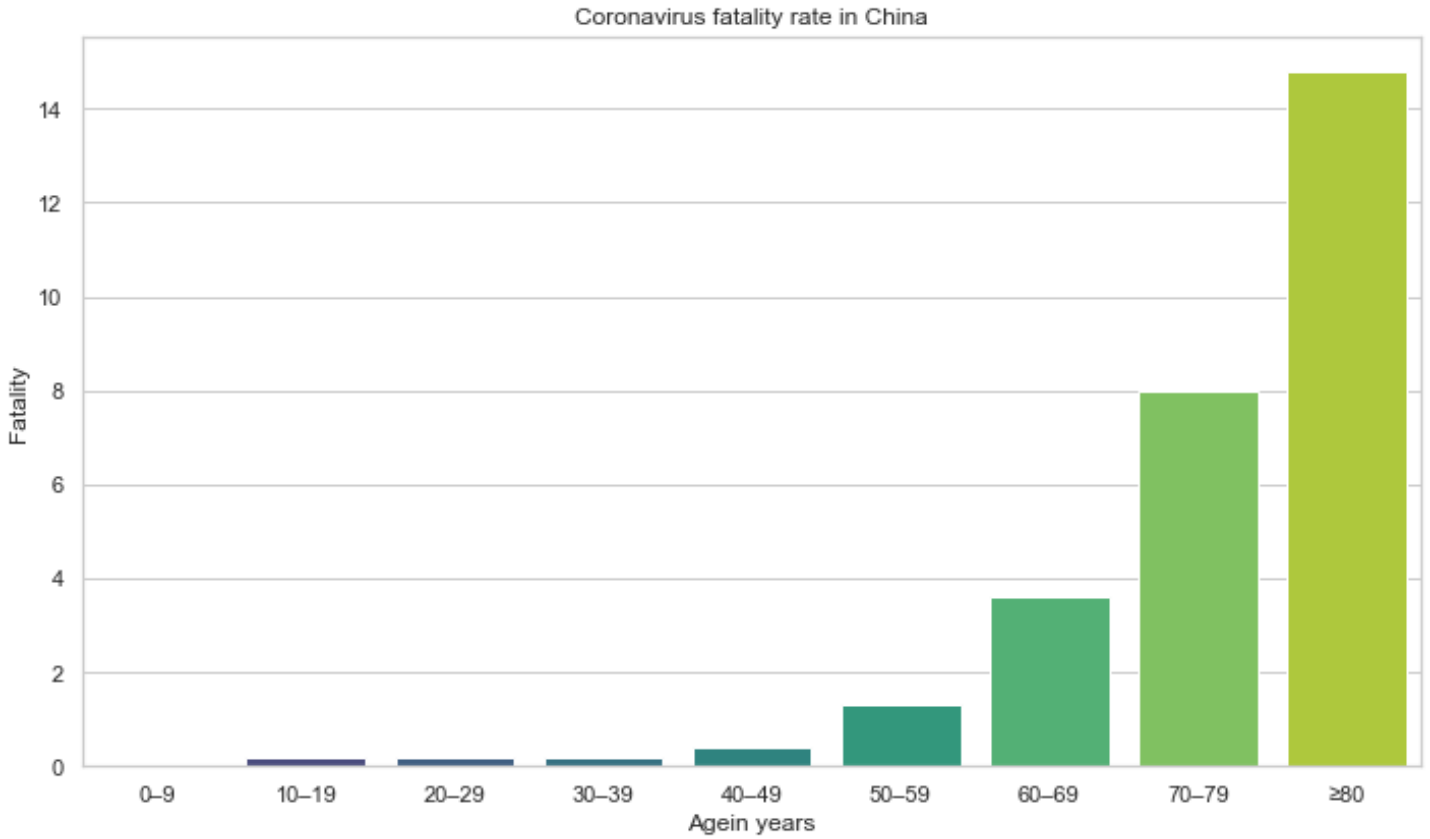


Image: Coronavirus fatality rate in China

Using Plotnine and Matplotlib:

```
# Importing relevant libraries
import matplotlib.pyplot as plt
import pandas as pd
from plotnine import ggplot, aes, geom_bar, labs, theme_minimal, theme, element_text


# Creating a DataFrame from the provided data
cases = {
    'Age': ['0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-69', '70-79', '≥80'],
    'Fatality': [0, 0.2, 0.2, 0.2, 0.4, 1.3, 3.6, 8, 14.8]
}
dataset = pd.DataFrame(cases)

# Create a bar plot using plotnine
plot = (
```

```
    ggplot(dataset, aes(x='Age', y='Fatality')) +
    geom_bar(stat='identity', fill='#1f78b4') +
    labs(title='Coronavirus fatality rate in China', x='Age in years') +
    theme_minimal() +
    theme(axis_text_x=element_text(angle=45, hjust=1))  # Rotating x-axis labels for
better readability
)

# Save the Plotnine plot to an image file
plot.save("coronavirus_fatality_rate_plot.png")

# Read the saved image file using Matplotlib and display it
img = plt.imread("coronavirus_fatality_rate_plot.png")
plt.figure(figsize=(10, 6))
plt.imshow(img)
plt.axis('off')
plt.show()
```

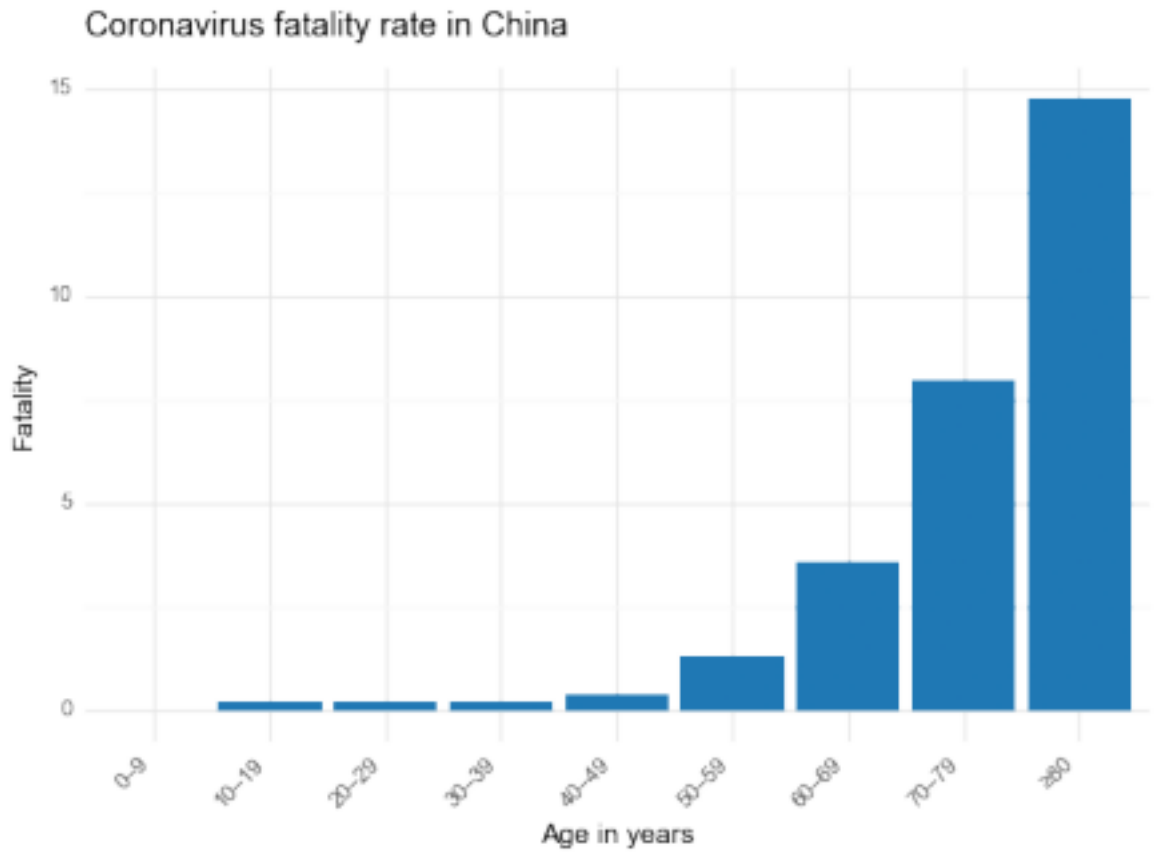


Image: Coronavirus fatality rate in China

Observations:

- **Youthful Resilience:**
  The age ranges of 0–9, 10–19, and 20–29 have exceptionally low death rates, highlighting how resilient younger populations are to adverse consequences.

- **Moderate Risk in Middle-Aged Groups:**
  In comparison to younger age groups, the 30-39 and 40-49 age groups have progressively higher mortality rates, indicating a moderate risk.

- **Elevated Risk in Elderly Populations:**
  Notable increases in mortality rates are noted in the age ranges of 50–59, 60–69, 70–79, and ≥80, underscoring the older population's increased susceptibility.

Implications for Decision Makers:

- **Targeted Vaccination Strategies:**
  Given the increased risk of severe consequences in older age groups, decision makers should give vaccine distribution top priority.

- **Resource Allocation:**
  To guarantee prompt and successful interventions, healthcare resources and capacities should be properly distributed, with an emphasis on older age groups.

Task 4. Data protection and data ethics (MLO3)

(a):

UK Electoral Commission hack exposed data of 40 million voters:

A cyberattack on the Electoral Commission in 2021 compromised the personal information of around 40 million UK voters. The security compromise was discovered in October 2022 and is thought to have been carried out by unknown "hostile actors." Although the specifics of what was accessed are unknown, the incident was swiftly reported to the appropriate UK agencies, and efforts were made to secure the Commission's system. While the Electoral Commission's data is said to be limited, US intelligence officials are concerned that it may be coupled with other publicly available data to generate extensive profiles of individuals. (TIMESOFINDIA.COM, 2023)

A "complex cyberattack" on the Electoral Commission compromised the personal information of around 40 million UK voters. The country's elections are overseen by the Electoral Commission, which revealed on Wednesday that it had identified suspicious activity on its network in October 2022. It was eventually determined, however, that unnamed "hostile actors" had obtained access to company systems more than a year earlier, in August 2021. According to the regulatory authority, the incident was spotted and reported within 72 hours to the Information Commissioner's Office (ICO) and the National Crime Agency. (TIMESOFINDIA.COM, 2023)

The details exposed because of the cyber incident are as follows (The Hacker News, n.d.):

- Name, first name, and surname
- Email addresses (personal and/or business)
- Home address if included in a webform or email
- Contact telephone number (personal and/or business)
- Content of the webform and email that may contain personal data
- Any personal images sent to the Commission.

- Home address in register entries
- Date on which a person achieves voting age that year

It's unclear why the disclosure was postponed for another ten months, but the Commission informed the BBC and The Guardian that it was done to prevent the adversary from gaining access, assess the nature of the breach, and implement security guardrails. (The Hacker News, n.d.)

The Effect on Real People:

The huge data breach at the UK's Electoral Commission exposes sensitive information such as names and addresses, potentially affecting millions. While the breach does not pose a large risk right away, it does open the door to unauthorised use, identity theft, and targeted phishing attacks. Individuals, particularly those who value their privacy, are now concerned about the protection of their personal data. The incident emphasises the significance of protecting electoral systems from cyber threats.

Beyond the immediate consequences, the breach has far-reaching consequences. Public trust in the electoral system may be jeopardised, raising concerns about security measures and potential weaknesses. To regain trust, the Commission must handle the aftermath by safeguarding systems and conducting investigations. Transparent communication is critical. This event serves as a sharp reminder of the persistent difficulty in protecting sensitive data, highlighting the continual work required to maintain trust in democratic processes.

(b):

1. GDPR Principles:
   a. Right to Control Personal Data:
      The GDPR emphasises individual's rights to their personal data by adopting principles such as the right to access, amend, and erase personal data. The exposed personal information, including names and addresses, raises GDPR issues in the context of the Electoral Commission's data breach. Individuals have the right to control their data, and any unauthorised access or use of such data is a violation of this fundamental principle.

   b. Right to Benefit from Personal Data:
      Individual's rights to benefit from their personal data are also recognised by the GDPR, particularly in terms of data portability and the right to be informed. The potential abuse of personal data in the context of the breach could infringe on individual's rights to benefit from their information. Unauthorised access to electoral records may have ramifications for individual's informational self-determination, even if it is not immediately monetizable.

2. UK Government Data Ethics Framework:

   a. Right to Control Personal Data:
      The UK Government Data Ethics Framework emphasises the importance of individual autonomy and transparency in data practises. The Electoral Commission's breach, which involved unauthorised access to voting rolls, raises ethical concerns. The Framework would urge strong measures to prevent such breaches and protect individual's right to data management.

   b. Right to Benefit from Personal Data:

Individual autonomy and transparency in data practises are emphasised in the UK Government Data Ethics Framework. The violation by the Electoral Commission, which entailed unlawful access to voter rolls, raises ethical concerns. The Framework would advocate for stringent safeguards to prevent such breaches and to protect individual's right to data management.

Conclusion:

While data ownership differs from physical asset ownership, both GDPR and the UK Government Data Ethics Framework emphasise the importance of individuals' rights in controlling and benefiting from their personal data. The principles highlight the importance of robust security measures, openness, and ethical considerations in the context of the Electoral Commission's breach to protect individuals' data rights and maintain trust in data operations. Even if data is not a tangible commodity, the breach creates serious ethical and legal implications because individuals' control and profits from their data are at stake.

References:

- TIMESOFINDIA.COM. (2023, August 9). UK Electoral Commission hack exposed data of 40 million voters. *The Times of India*. https://timesofindia.indiatimes.com/gadgets-news/uk-electoral-commission-hack-exposed-data-of-40-million-voters/articleshow/102576780.cms
- The Hacker News. (n.d.). *U.K. Electoral Commission breach exposes voter data of 40 million Britons*. https://thehackernews.com/2023/08/uk-electoral-commission-breach-exposes.html
- *ESA: CG: LI: FLOW: F.33111, Sterling treasury bills - Office for National Statistics*. (2023, November 21). https://www.ons.gov.uk/economy/grossdomesticproductgdp/timeseries/navg/pusf
- *DIM: LA dividend receipts - seasonally adjusted - Office for National Statistics*. (2023, September 28). https://www.ons.gov.uk/economy/grossdomesticproductgdp/timeseries/fdfs/ukea
- Author: OpenAI's GPT-3 model

  Title: Autocorrelation Calculation in Python
- Weekly, C. C. (2020). The epidemiological characteristics of an outbreak of 2019 novel coronavirus diseases (COVID-19) — China, 2020. *China CDC Weekly*, *2*(8), 113–122. https://doi.org/10.46234/ccdcw2020.032

- BBC News. (2020, February 18). Coronavirus: Largest study suggests elderly and sick are most at risk. *BBC News*. https://www.bbc.co.uk/news/world-asia-china-51540981