|  |  |
| --- | --- |
| Blue letters on a white background  Description automatically generated | Text  Description automatically generated with medium confidence |

# Student Assessment Submission and Declaration

When submitting evidence for assessment, you must sign a declaration confirming that the work is your own.

|  |  |  |  |
| --- | --- | --- | --- |
| Student name: |  | ESL ID No |  |
| BSU ID No |  |
| Submission date: | | |  |
|  | | |  |
| Programme: |  | | |
| Module name and code: | Software Foundation CPUF001 | | |
| Title: |  | | |
| Assessor name: |  | | |

**Plagiarism**

Plagiarism is a form of cheating. Plagiarism must be avoided at all costs and students who break the rules, however innocently, may be penalised. It is your responsibility to ensure that you understand correct referencing practices. As a university level student, you are expected to use appropriate references throughout and keep carefully detailed notes of all your sources of materials for material you have used in your work, including any material downloaded from the Internet. Please consult the relevant unit lecturer or your course tutor if you need any further advice.

|  |  |  |  |
| --- | --- | --- | --- |
| **Student declaration**  I certify that the assignment submission is entirely my own work. I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice. | | | |
| Student signature: |  | Date: |  |

**Table of Contents**

[Student Assessment Submission and Declaration 1](#_Toc198568528)

[1. Introduction 3](#_Toc198568529)

[2. Solution design 3](#_Toc198568530)

[2.1 Flowchart and Pseudocode overview 3](#_Toc198568531)

[2.2 Program walkthrough 6](#_Toc198568532)

[2.3 Script walkthrough 7](#_Toc198568533)

[2.4 Technical breakdown of code 7](#_Toc198568534)

[3. Reflective evaluation 9](#_Toc198568535)

[References 11](#_Toc198568536)

[Appendix 12](#_Toc198568537)

# 1. Introduction

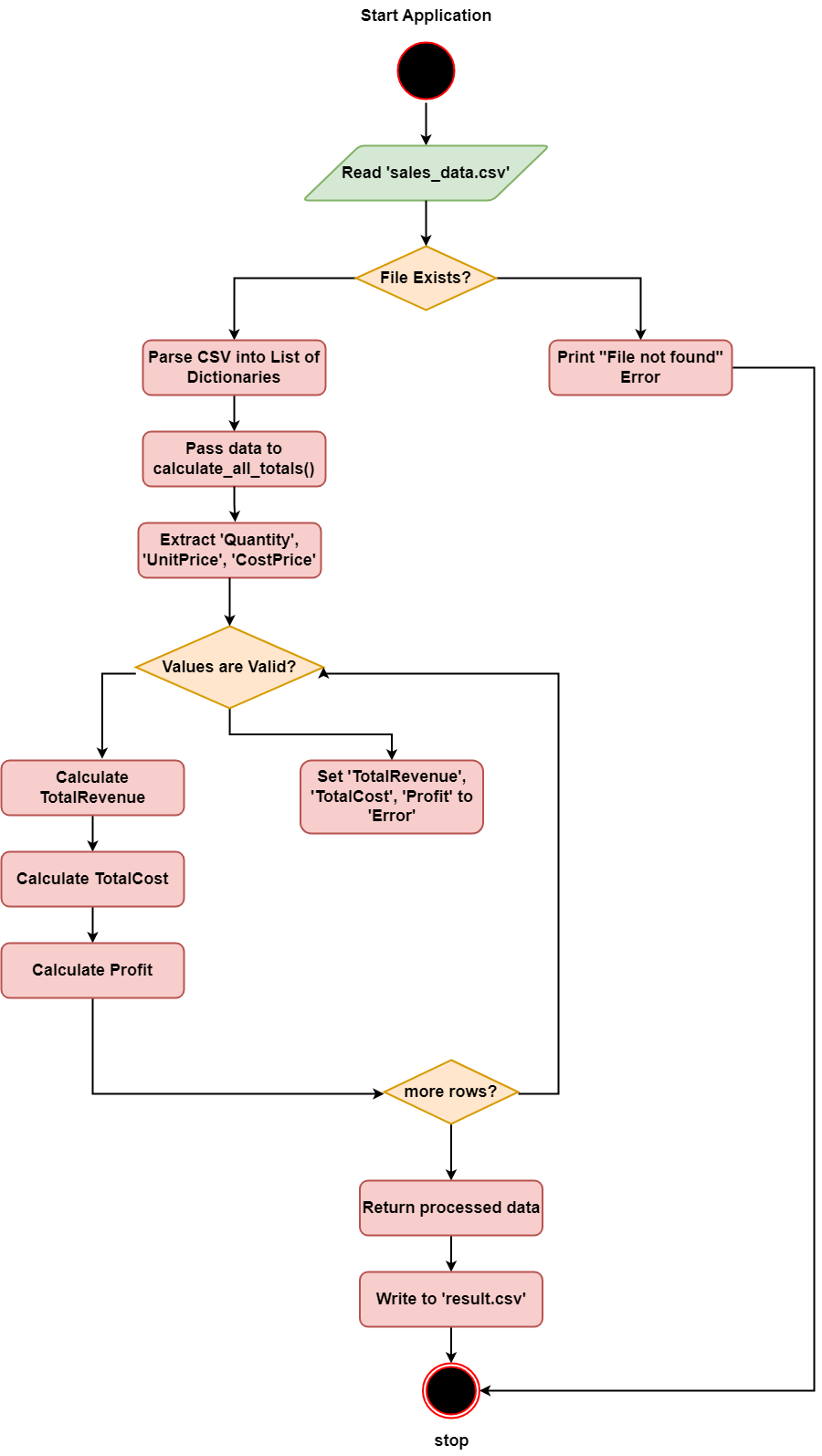
Development of this project has been directed by an organized study of what must happen to automate sales data and produce the desired output. The problem is finding a way to use tools to calculate total revenue, total expenses, and the profit from sales, and also record and organize the data in a structured output file. It eliminates the problem of handling sales data by manual methods, which leads to errors. Modular scripts have been used for the solution, allowing it to be used many times, understood easily, and maintained conveniently. There has been special effort made to ensure the revenue, cost, and profit amounts are correct from the sales records. Exception handling and clear separation of responsibilities have been used to develop the code effectively. All the source code for this project has been placed on GitHub for everyone to access and see.

***Github repository link:*** <https://github.com/Tufayel578/Sales-data-analysis/tree/main>

# 2. Solution design

## 2.1 Flowchart and Pseudocode overview

***Flowchart***



**Figure 1: Application flowchart**

The sales analysis program is explained through a flowchart that reflects its logic using traditional UML symbols. An oval marks where the process starts and ends, a rectangle stands for an action, while a diamond is used for decision points. The flow between each step is shown by using arrows. The first step is for the application to try reading the sales\_data.csv file. Records are then handled one at a time: you enter the quantity, determine its unit and cost prices, confirm their validity, and calculate the revenue, expenses, and profit. Newly calculated values are adjusted to an appropriate format, and kept mistaken or missing values are not accepted and marked with an error.

***Pseudocode***

|  |
| --- |
| Algorithm: Sales Data Financial Analysis  Input:  CSV file sales\_data.csv containing sales records with fields: Quantity, UnitPrice, and CostPrice.  Output:  CSV file result.csv containing original data plus calculated fields: TotalRevenue, TotalCost, and Profit.  Procedure:  1. Start  2. Read CSV file sales\_data.csv   * If the file does not exist, print an error message and terminate.   3. For each record in the CSV data:   * Extract Quantity, UnitPrice, CostPrice values * Attempt to convert Quantity to integer, UnitPrice, and CostPrice to float * If conversion fails, set TotalRevenue, TotalCost, and Profit to "Error" for that record * Else:  1. Calculate TotalRevenue = Quantity × UnitPrice 2. Calculate TotalCost = Quantity × CostPrice\ 3. Calculate Profit = TotalRevenue − TotalCost 4. Round all calculated values to two decimal places 5. Store results in the record   4. Write all processed records to the output CSV file result.csv  5. Print the success message  6. End |

## 2.2 Program walkthrough

The main and calculation Python files are defined to handle different tasks in the pipeline used for processing data with a CLI. It uses the data from the CSV file (sales\_data.csv), calculates financial metrics, and produces output in a new file (result.csv).

***main.py***

The file directs the workflow of the application and has the main functions described below.

**read\_csv():** The function uses the *csv.DictReader* library to read input values from sales\_data.csv.

**write\_csv(data):** This step saves the data to the result.csv file. It employs *csv.DictWriter* helps organize data so that the headers in rows mirror the processed information (Edastama *et al*., 2021).

**main():** The part of the script that propels the film. The first action is to use *read\_csv()* to bring in the data, send it to the calculation module (*calculate\_all\_totals()* in *calculation.py*), and lastly use *write\_csv()* to generate the outcomes (Kamil *et al*., 2023).

***calculation.py***

The purpose of the calculation.py module is to perform all necessary computing processes on the sales data.

**Function Used:** The function goes through every row in the dataset and calculates all the financial values.

The calculation program is developed using three calculations. The formula of these calculations are,

*TotalRevenue = Quantity × UnitPrice*

*TotalCost = Quantity × CostPrice*

*Profit = TotalRevenue - TotalCost*

All rows are examined and turned into numbers so that numerical activities can be used during iteration (Wengler *et al*., 2021). The try-except structure is used to manage data that is missing or damaged.

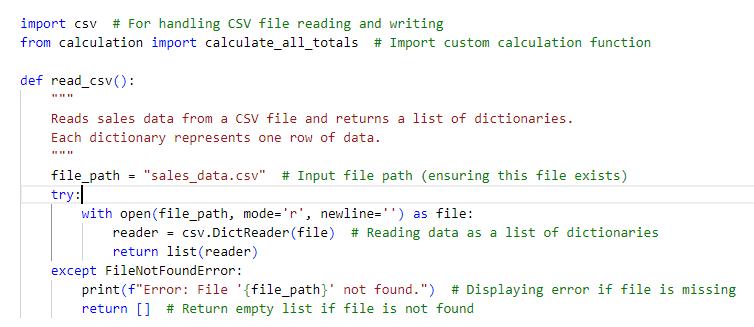
## 2.3 Script walkthrough

The file sales\_data.csv is used by *main.py*, but the *\_\_main\_\_ block* allows it to be executed from the command line. The walkthrough of the script file:

* The Windows Batch Script (run\_analysis.bat) has been made to perform the analysis with fewer steps.
* The script allows even non-technical users to interact with Python by clicking on a simple file that contains the command.
* The combination of CLI and scripting makes it easy to perform similar operations on data, no matter the platform (Rachmad, and Budiyanto, 2022).

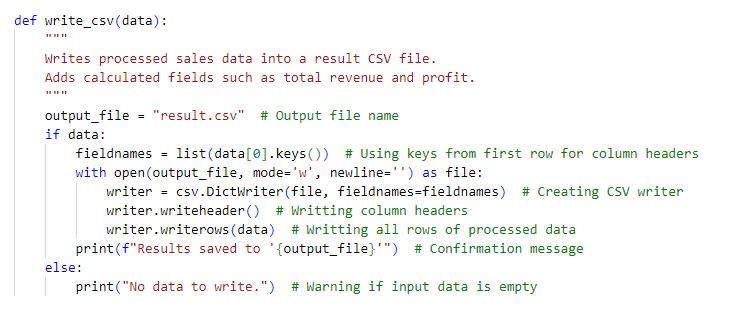
## 2.4 Technical breakdown of code

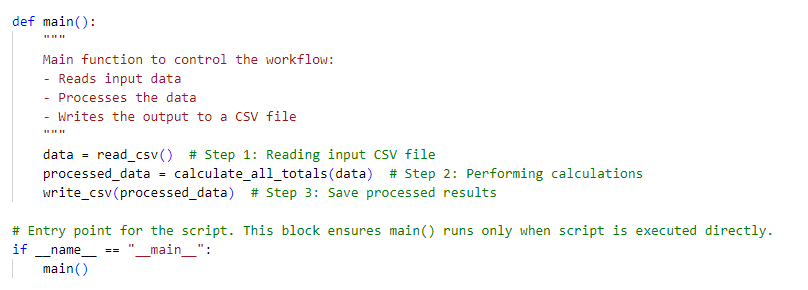
***Execution of main.py***



**Figure 2: Loading sales data and defining read.csv**

The read\_csv() function easily reads the sales data from a CSV file and returns it as a list of dictionaries. It deals with handling missing files properly, ensuring all data used in the next process is accurate.

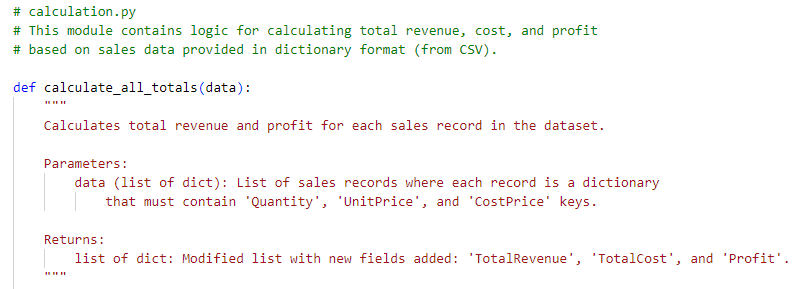


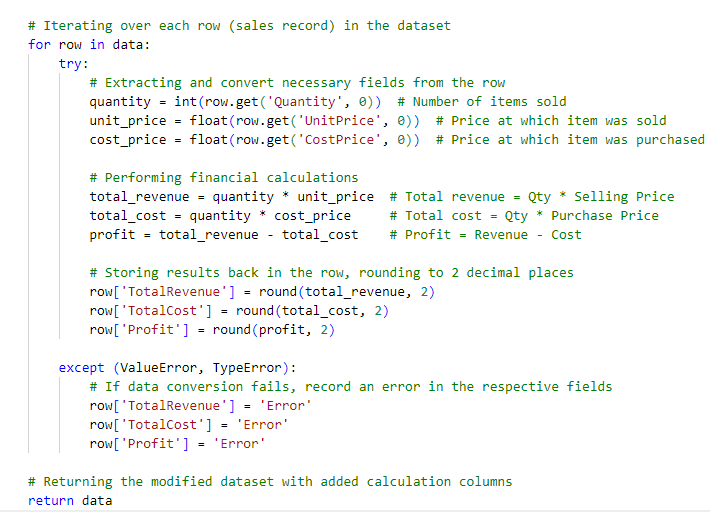


**Figure 3: Defining main.csv and write.csv**

Processed sales data and the newly computed financial figures are recorded into a new CSV file by using write\_csv(). The main() function handles the entire process of reading, calculating, and writing data.

***Execution of calculation.py***





**Figure 4: Defining all total and calculation implementation**

The module determines the total revenue, total cost, and profit for every record by processing sales data. It provides essential financial information, making it easier to analyze and track the company’s sales.

# 3. Reflective evaluation

The project experience has given me a better understanding of modular programming and working with commands. Moving the logic to calculation.py has helped me understand how to structure the code so that various parts can be separated and updated easily. Handling real data has improved my ability to use the CSV module. The use of financial statements for Total Revenue, Total Cost, and Profit has allowed me to become more familiar with data-based calculations. I believe that by using try-except blocks, I can ensure that the program continues to operate even when receiving data that is incomplete or incorrect. Thanks to CLI-enabled scripts, I can develop tools that work smoothly for any purpose, without an interface. Moreover, working on a batch script has helped me use basic scripting steps to support automation and better system integration.

I have spotted several ways in which improvements can be made. Example uses include employing more reliable validation and also using pandas libraries to enhance the system’s performance. I discovered that using automated tests is very useful and plan to utilize Python’s testing frameworks to unit test final projects. I have come to value the importance of planning, outlining and talking clearly after making a report and flowchart. The experience from the project has improved the way I use and analyze data.

# References

Edastama, P., Bist, A.S. and Prambudi, A., 2021. Implementation of data mining on glasses sales using the apriori algorithm. *International Journal of Cyber and IT Service Management*, *1*(2), pp.159-172.

Kamil, H., Mukhlis, M. and Bachtiar, Y., 2023. Integration of ANP and TOPSIS Methods in Prioritizing Sales Strategies for Frozen Food Products. *Jurnal Riset Ilmu Teknik*, *1*(2), pp.102-114.

Rachmad, Y.E. and Budiyanto, B., 2022, March. Perception Analysis of Sales Volume on Partner Who Using Three Food Delivery Apps in Surabaya. In *International Conference of Business and Social Sciences* (pp. 116-122).

Wengler, S., Hildmann, G. and Vossebein, U., 2021. Digital transformation in sales as an evolving process. *Journal of Business & Industrial Marketing*, *36*(4), pp.599-614.

# 

# Appendix

***Code of main.py***

|  |
| --- |
| # main.py  # This script reads sales data from a CSV file, processes the data to calculate totals,  # and writes the results into a new CSV file. It is designed to be executed from the CLI.  import csv # For handling CSV file reading and writing  from calculation import calculate\_all\_totals # Import custom calculation function  def read\_csv():  """  Reads sales data from a CSV file and returns a list of dictionaries.  Each dictionary represents one row of data.  """  file\_path = "sales\_data.csv" # Input file path (ensuring this file exists)  try:  with open(file\_path, mode='r', newline='') as file:  reader = csv.DictReader(file) # Reading data as a list of dictionaries  return list(reader)  except FileNotFoundError:  print(f"Error: File '{file\_path}' not found.") # Displaying error if file is missing  return [] # Return empty list if file is not found  def write\_csv(data):  """  Writes processed sales data into a result CSV file.  Adds calculated fields such as total revenue and profit.  """  output\_file = "result.csv" # Output file name  if data:  fieldnames = list(data[0].keys()) # Using keys from first row for column headers  with open(output\_file, mode='w', newline='') as file:  writer = csv.DictWriter(file, fieldnames=fieldnames) # Creating CSV writer  writer.writeheader() # Writting column headers  writer.writerows(data) # Writting all rows of processed data  print(f"Results saved to '{output\_file}'") # Confirmation message  else:  print("No data to write.") # Warning if input data is empty  def main():  """  Main function to control the workflow:  - Reads input data  - Processes the data  - Writes the output to a CSV file  """  data = read\_csv() # Step 1: Reading input CSV file  processed\_data = calculate\_all\_totals(data) # Step 2: Performing calculations  write\_csv(processed\_data) # Step 3: Save processed results  # Entry point for the script. This block ensures main() runs only when script is executed directly.  if \_\_name\_\_ == "\_\_main\_\_":  main() |

***Code of calculation.py***

|  |
| --- |
| # calculation.py  # This module contains logic for calculating total revenue, cost, and profit  # based on sales data provided in dictionary format (from CSV).  def calculate\_all\_totals(data):  """  Calculates total revenue and profit for each sales record in the dataset.    Parameters:  data (list of dict): List of sales records where each record is a dictionary  that must contain 'Quantity', 'UnitPrice', and 'CostPrice' keys.  Returns:  list of dict: Modified list with new fields added: 'TotalRevenue', 'TotalCost', and 'Profit'.  """  # Iterating over each row (sales record) in the dataset  for row in data:  try:  # Extracting and convert necessary fields from the row  quantity = int(row.get('Quantity', 0)) # Number of items sold  unit\_price = float(row.get('UnitPrice', 0)) # Price at which item was sold  cost\_price = float(row.get('CostPrice', 0)) # Price at which item was purchased  # Performing financial calculations  total\_revenue = quantity \* unit\_price # Total revenue = Qty \* Selling Price  total\_cost = quantity \* cost\_price # Total cost = Qty \* Purchase Price  profit = total\_revenue - total\_cost # Profit = Revenue - Cost  # Storing results back in the row, rounding to 2 decimal places  row['TotalRevenue'] = round(total\_revenue, 2)  row['TotalCost'] = round(total\_cost, 2)  row['Profit'] = round(profit, 2)  except (ValueError, TypeError):  # If data conversion fails, record an error in the respective fields  row['TotalRevenue'] = 'Error'  row['TotalCost'] = 'Error'  row['Profit'] = 'Error'  # Returning the modified dataset with added calculation columns  return data |