I. Executive Summary

The program developed in this project serves the needs of Transport for Wales (TfW) in conducting a comprehensive survey at Aberystwyth to understand passenger preferences and track the sale of train tickets between Aberystwyth and various cities. The survey focuses on seven key cities: Birmingham, Cardiff, Hereford, Liverpool, Manchester, Newport, and Shrewsbury, each represented by a unique symbol ('B', 'C', 'H', 'L', 'M', 'N', 'S').

The program is designed to address the following tasks:

1. **Calculate the percentage of Birmingham and Manchester ticket sales in a given cities string.**
2. **Calculate the total sale value based on ticket prices provided in Table 1.**
3. **Convert a cities string into a dictionary containing the symbol and count of each city.**
4. **Record and analyse train ticket sales data, including counting the number of tickets sold for each city, determining the number of hours in which tickets were sold, and calculating the total sales.**

These tasks are essential for TfW to gain insights into passenger preferences, sales trends, and ticket distribution among the seven cities. The program ensures accuracy and efficiency in data analysis and reporting.

II. Technical Overview

The implementation choices in the program reflect my commitment to meeting TfW's requirements effectively. I have employed functions and data structures to enhance code organization, readability, and reusability. Here's an overview of the technical approach:

Task 1: To calculate percentage function counts the occurrences of 'B'& ‘M’ (Birmingham and Manchester) in the cities string and calculates the percentage of Birmingham and Manchester sales.

1) **Function Definition:**

   According to the definition of a Function, they are reusable blocks of code that perform a specific task. In the code, I started by defining a function called “calculate percentage”,

which takes a parameter called “cities string”. This is designed to allow the user to apply the percentage calculation to different city names.

2) **Counting Occurrences**:

   The next step, I utilized the count () method to count the occurrences of each city string. The count is then stored in variables ‘count Birmingham’ and ‘count Manchester’.

3) **Total Cities Count**:

   Here we calculated the percentage, to know the total number of cities in the string, The city string is then broken down into a list of individual cities, and ‘Len ()’ was used to find the list length.

4) **Calculating Percentage**:

   Now that we have the counts and the total number of cities, the code proceeds to calculate the percentage of occurrences for each city. This involves dividing the count of each city by the total number of cities and multiplying by 100 (so therefor, each city percentage are computed by dividing the count of each city by the total number of cities and multiplying by 100)

5) Returning Results:

   Here the function returns the percentage calculated as a tuple, the calculated percentages are printed to the console with a formatted string, displaying the results.

Task 2: The calculate total sale function uses a dictionary of city prices to calculate the total sale for a given cities string. The cities are represented by single-letter symbols. The functions take a single argument.

1) I defined a dictionary called ‘city_prices’, mapping each city symbol to its corresponding price

2) The sales total is then calculated, summing up the cities price in the ‘cities_ string’ using comprehension list

3)The total sale is then rounded to two decimal places using ‘round’ function

4) The rounded sale total is returned based on the price of the cities

Task 3: The cities _to_ dictionary function converts a cities string into a dictionary containing city symbol counts. I utilized a dictionary to store these counts efficiently.

1)On this task, I initialized as an empty dictionary called ‘city _ counts’, to store the counts of each city
2)The function iterates through each character in the ‘cities _ string’.
3)The code then checks if the city is already a key in the ‘city _counts’ dictionary
4)if the city is already in the dictionary, it increases the count for that city
5)Else, if the city is not in the dictionary, it adds the city as a key with a count of 1
6)Finally, the function returns the resulting dictionary containing the counts of each unique city symbol in the input string.

Task 4: The record _ and_ analyse _ sales function enables the continuous input of cities strings until the user chooses to exit by typing 'quit'. I validate the input to ensure that only recognized city symbols are accepted. The program records each cities string in a file named with the current date and accumulates data for analysis.

The function ‘record _ and _ analyse _ sales’ is designed to manage train ticket sales, analyse the data, and provide a summary.

1) Initialization: In the list ‘all _ cities _ data’ cities data entered by the user are stored variable ‘total _ sales’ is initialized to zero, which accumulate the total sales amount

2) Data Entry Loop:

- An infinite loop is entered by the function using While True
- The user is prompt to enter a cities string or type quit.
- If the user inputs ‘quit’ the data entry phase ends (loop breaks)

3) Validation Input:

-- The entered city symbols are checked if its valid (B, C, H, L, M, N, S)
-- If a symbol that’s invalid is noticed, an error message is printed and continues to the next iteration, prompting the user to enter the cities string again

4) Sales Record:

-- ‘datetime.date.today()’ obtains current data and formatted as "%Y-%m-%d
-- The cities string is appended to a file named with the current date and a ".txt" extension and saved in the current directory in which the python file is saved.

5) Accumulation of Data:
-- The entered cities string is added to the ‘all _ cities _ data’ list.

-- The function ‘calculate _ total _ sale’ calculates the total sale for the entered cities string, and the result is added to ‘total _ sales’

6) Analysis:
-- Dictionary ‘city _ counts ‘and dictionary ‘hours _ sold’ are initialized to store the count of each city and the hours of sales for each city
-- For each cities string in ‘all _ cities _data’, function ‘cities_ to_ dictionary’ gets the counts of each city in the string and the total hours (1) is assigned to total hours.
-- Nested loops iterate through the cities and update the counts in city_counts and accumulate hours in hours _ sold.

7) Summary:

The function prints the number of tickets sold for each city, the number of hours in which tickets of each city have been sold.
8) Execution
- The script checks if it's being run directly (if __name__ == "__main__":) and if so, calls the record _ and _ analyse _ sales function.

-

III. Software Testing

Testing is a critical aspect of ensuring the program's correctness and reliability. I have rigorously tested the program at multiple levels:

Unit Testing: Each function was tested individually to ensure that it correctly performs its specific task. I used a variety of test cases to validate the functions' accuracy.

Integration Testing: The entire program was tested to ensure that the different components work together seamlessly. I verified that data flows correctly between functions and that the program handles user input and file operations effectively.

Functional Testing: I tested the program, simulating real-world usage by entering cities strings and analysing the output. This testing approach helped us verify that the program meets TfW's requirements.

Also,

1) Verified that the function returned the expected results.

2) Test the code with various scenarios such as using empty city names that are not present in the string

3) Testing with large datasets to ensure performance

4) Ensure the code is user-friendly and provides meaningful results.

IV. Reflections and Future Work

My code, while functional, can be further improved and extended to better serve TfW's needs. Here are some areas of reflection and potential future work:

User Interface: Enhancing the user interface with a more user-friendly input and feedback system could make the program more accessible to TfW staff.

Data Visualization: Implementing data visualization tools could assist TfW in understanding trends and patterns in ticket sales more effectively.

Error Handling: Further improvements in error handling can be made to provide more informative error messages to users.

Optimization: I can explore ways to optimize the code for performance, especially when dealing with larger datasets.

User Documentation: Creating user documentation and guides can help TfW staff effectively use the program.

In total, the project took approximately more than 40 hours to complete, including coding, testing, and documentation.

V. Appendix

Conclusion

In conclusion, the program developed for TfW successfully addresses the tasks outlined in the project requirements. It provides valuable insights into passenger preferences, ticket sales, and the distribution of sales among the seven cities of interest. The rigorous testing and thoughtful design ensure the reliability of the program. Further enhancements and refinements can be explored to maximize the program's utility for TfW.

This report presents a comprehensive overview of my work, including the technical choices made, testing procedures, and considerations for future improvements. The program is a valuable tool for TfW in managing and analyzing train ticket sales data.