System Requirements Specification Index

For

Data Analysis Tool Console Application

Version 1.0

IIHT Pvt. Ltd.

fullstack@iiht.com

TABLE OF CONTENTS

Data Analysis Tool Console

System Requirements Specification

1 PROJECT ABSTRACT

SalesMetrics Solutions requires a comprehensive data processing application to analyze sales performance metrics across multiple regions and product categories. The company needs a Python console application that can efficiently process large volumes of sales data, calculate key performance statistics, visualize trends, and generate actionable business insights. This tool will help sales managers identify growth opportunities and make data-driven decisions to optimize their product portfolio and regional sales strategies.

2 BUSINESS REQUIREMENTS:

Screen Name	Console input screen
Problem Statement	1. Application must process sales data by region and product 2. System must calculate basic statistics (totals, averages) 3. Program must identify top-performing products and regions

3 CONSTRAINTS

3.1 INPUT REQUIREMENTS

1. Data Structure:
   - Sales data organized by regions and products
   - Data provided as a nested dictionary
   - Example: sales_data[region][product] = quantity_sold

3.2 CALCULATION CONSTRAINTS

1. Regional Analysis
   - Calculate total sales per region
   - Identify highest and lowest performing regions
2. Product Analysis:
   - Calculate total sales per product across all regions
   - Identify top and bottom products

3.3 OUTPUT CONSTRAINTS

1. Display Format:
   - Show "Sales Data Analysis Tool"
   - Show analysis type and results
   - Format numbers appropriately

4. TEMPLATE CODE STRUCTURE:

1. Analysis Functions:
   - analyze_regional_sales(sales_data)
   - analyze_product_performance(sales_data)

- display_results(results, analysis_type)

2. Main Section:

- load_sales_data()
- display_menu()
- perform_selected_analysis(analysis_type)

5. DETAILED FUNCTION IMPLEMENTATION GUIDE

5.1 Core Analysis Functions

1. Write a Python function to analyze sales performance by region. Define: `analyze_regional_sales(sales_data)`

The function should:

- Accept one parameter: `sales_data` (nested dictionary structure)
- Validate that input is not None: `if sales_data is None:`
- Raise `TypeError` with message "sales_data cannot be None" for None input
- Initialize empty dictionary for results: `regional_analysis = {}`
- Initialize tracking variables for highest and lowest regions:
  - `highest_region = {"name": "", "total": 0}`
  - `lowest_region = {"name": "", "total": float('inf')}`
- Use **outer for loop** to iterate through regions: `for region, products in sales_data.items():`
- Initialize regional total: `regional_total = 0`
- Use **nested for loop** to iterate through products: `for product, amount in products.items():`
- Validate each sales amount:
  - Check type: `if not isinstance(amount, (int, float)):`
  - Raise `TypeError` with message "Sales amount for {region}, {product} must be a number"
  - Check for negative values: `if amount < 0:`
  - Raise `ValueError` with message "Sales amount for {region}, {product} cannot be negative"
- Accumulate regional total: `regional_total += amount`

- Track highest performing region: `if regional_total > highest_region["total"]:`
- Track lowest performing region: `if regional_total < lowest_region["total"]:`
- Store regional total: `regional_analysis[region] = regional_total`
- After processing all regions, add performance labels:
  - Highest region: `(total, "Highest performing region")`
  - Lowest region: `(total, "Lowest performing region")`
  - Other regions: `(total, "")`
- Return dictionary with format: `{region: (total, label), ...}`
- Example: analyze_regional_sales(sample_data) should return {"North": (250, "Highest performing region"), ...}

2. Write a Python function to analyze product performance across all regions. Define: `analyze_product_performance(sales_data)`

The function should:

- Accept one parameter: `sales_data` (nested dictionary structure)
- Validate that input is not None: `if sales_data is None:`
- Raise `TypeError` with message "sales_data cannot be None" for None input
- Initialize empty dictionary for results: `product_analysis = {}`
- First, collect all unique products using a set: `all_products = set()`
- Use **first loop** to gather all product names: `for region, products in sales_data.items():`
- Add products to set: `for product in products: all_products.add(product)`
- Use **second loop** to calculate totals for each product: `for product in all_products:`
- Initialize product total: `product_total = 0`
- Use **nested loop** to sum across regions: `for region, products in sales_data.items():`
- Check if product exists in region: `if product in products:`
- Add to total: `product_total += products[product]`
- Store product total: `product_analysis[product] = product_total`

- Sort products by total sales: `sorted_products = sorted(product_analysis.items(), key=lambda x: x[1], reverse=True)`
- Create result dictionary with rankings:
    - Top product (index 0): `(total, "Top product")`
    - Bottom product (last index): `(total, "Bottom product")`
    - Middle products: `(total, "")`
- Return dictionary with format: `{product: (total, label), ...}`
- Example: analyze_product_performance(sample_data) should return {"Product A": (450, "Top product"), ...}

5.2 Display and Data Functions

3. Write a Python function to display analysis results in formatted tables. Define: `display_results(results, analysis_type)`

The function should:

- Accept two parameters: `results` (dictionary) and `analysis_type` (string)
- Print header: `print("\nSales Data Analysis Tool")`
- Print analysis type: `print(f"Analysis Type: {analysis_type}")`
- Print separator: `print("-" * 50)`
- Use conditional logic to format output based on analysis type:
- **For Regional Sales Analysis:**
    - Print column headers: `print(f"{'Region':<10}{'Total Sales':^15}{'Performance':>20}")`
    - Print separator: `print("-" * 50)`
    - Loop through results: `for region, (total, label) in results.items():`
    - Format output: `print(f"{region:<10}${total:>14,.2f}{label:>20}")`
- **For Product Performance Analysis:**
    - Print column headers: `print(f"{'Product':<10}{'Total Units':^15}{'Ranking':>20}")`
    - Print separator: `print("-" * 50)`
    - Loop through results: `for product, (total, label) in results.items():`

    - Format output: `print(f"{product:<10}{total:>14,.0f}{label:>20}")`
- Print final separator: `print("-" * 50)`
- Handle empty results gracefully
- Use proper string formatting for alignment and number formatting
- Example: Function should produce well-formatted tabular output with proper alignment

4. Write a Python function to load sample sales data. Define: `load_sales_data()`

The function should:

- Take no parameters
- Return a nested dictionary representing sales data structure
- Use the following data structure format:

```
sales_data = {
    "North": {
        "Product A": 120,
        "Product B": 85,
        "Product C": 45
    },
    "South": {
        "Product A": 95,
        "Product B": 110,
        "Product C": 30
    },
    "East": {
        "Product A": 105,
        "Product B": 90,
        "Product C": 40
    },
    "West": {
        "Product A": 130,
        "Product B": 120,
        "Product C": 50
    }
}
```

- Return the complete sales_data dictionary
- Ensure all sales amounts are positive numbers (int or float)

- Maintain consistent product names across all regions
- Example: load_sales_data() should return the complete nested dictionary structure

5.3 Main Program Function

5. Write a Python main program to demonstrate the data analysis tool. Define: `main()` function

The function should:

- Call `load_sales_data()` to get the data: `data = load_sales_data()`
- Use infinite loop for menu interaction: `while True:`
- Display menu options:
  - `print("\nSales Data Analysis Tool")`
  - `print("1. Regional Sales Analysis")`
  - `print("2. Product Performance Analysis")`
  - `print("3. Exit")`
- Get user choice: `choice = int(input("\nEnter your choice (1-3): "))`
- Use try-except block to handle input errors
- **For choice 1 (Regional Analysis):**
  - Call `results = analyze_regional_sales(data)`
  - Call `display_results(results, "Regional Sales Analysis")`
- **For choice 2 (Product Analysis):**
  - Call `results = analyze_product_performance(data)`
  - Call `display_results(results, "Product Performance Analysis")`
- **For choice 3 (Exit):**
  - Print "Thank you for using the Sales Data Analysis Tool!"
  - Break from loop
- **For invalid choices:**
  - Print "Invalid choice. Please enter a number between 1 and 3."
- **Handle ValueError for non-numeric input:**
  - Print "Invalid input. Please enter a number."
- **Handle general exceptions:**
  - Print `f"An error occurred: {str(e)}"`
- Include `if __name__ == "__main__": main()` at the end
- Example: Program should provide continuous menu-driven interaction until user exits

5.4 Implementation Requirements

Key Technical Requirements:

Nested Loop Structure:

- **analyze_regional_sales()** must use outer loop for regions and inner loop for products
- **analyze_product_performance()** must use multiple loops: one to collect products, another to calculate totals
- Both functions must demonstrate proper nested iteration through the data structure

Error Handling:

- All analysis functions must validate input parameters (None checks)
- Type validation for sales amounts (must be int or float)
- Value validation for sales amounts (must be non-negative for regional analysis)
- Proper exception raising with descriptive messages

Data Structure Management:

- Input: Nested dictionary `{region: {product: amount, ...}, ...}`
- Output: Dictionary with tuple values `{item: (total, label), ...}`
- Proper handling of missing products in some regions
- Efficient data aggregation and sorting

Formatting Requirements:

- Use f-string formatting for all output
- Proper column alignment (left, center, right alignment)
- Number formatting with commas and appropriate decimal places
- Consistent table structure with headers and separators

Performance Tracking:

- Identify highest/lowest performing regions with exact labels
- Identify top/bottom performing products with exact labels
- Handle edge cases (single region, single product, empty data)
- Maintain data integrity throughout processing

5.5 Testing Considerations

Expected Behaviors:

- Empty input data should return empty results dictionary
- Single region/product should be handled gracefully
- Boundary values (zero sales, very large numbers) should process correctly
- Invalid data types should raise appropriate exceptions
- Menu system should handle invalid inputs and continue running
- All calculations should be mathematically accurate
- Output formatting should be consistent and professional

Integration Requirements:

- All functions must work together seamlessly in the main program
- Data flow from load_sales_data() → analysis functions → display_results()
- Exception handling should not crash the main program
- Menu system should allow repeated analysis without restarting

6. EXECUTION STEPS TO FOLLOW:

1. Run the program

2. Select analysis type from menu

3. View calculated statistics

4. Repeat with different analysis or exit

Execution Steps to Follow:

- All actions like build, compile, running application, running test cases will be through Command Terminal.

- To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal

- This editor Auto Saves the code

- If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page)

- These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.

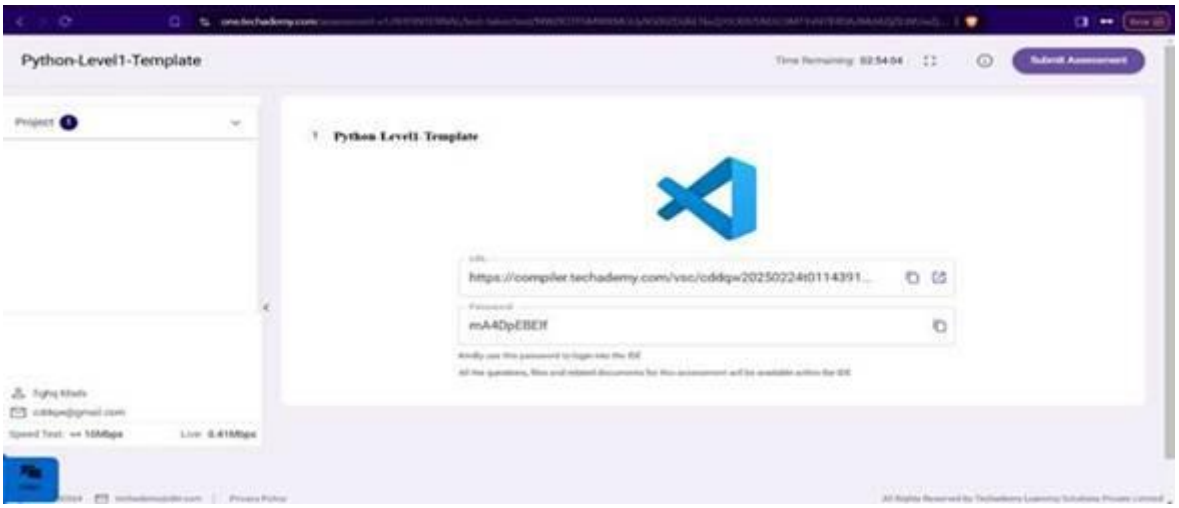
- To launch application: python3 filename.py
- To run Test cases: python3 -m unittest

Screen shot to run the program

To run the application

Python3 filename.py

To run the testcase python -m unittest



- Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on “Submit Assessment” after you are done with code.