

System Requirements

Specification Index

For

Merging Employee Data from Multiple Sources for Analysis

(Topic: Merging and Joining DataFrames)

Version 1.0

Employee Data Analysis Console

Project Abstract

The Employee Data Analysis system is a Python-based tool developed to merge and analyze employee data and salary information. The system processes two separate CSV files, one containing employee information (including Employee_ID, Name, and Department) and the other containing salary details (Salary and Employee_ID). By merging these datasets, the system enables the calculation of the average salary per department and provides the ability to export the processed data into an Excel file for reporting purposes. This tool simplifies employee salary analysis and enhances the decision-making process related to employee compensation within organizations.

Business Requirements:

- The system should load employee and salary data from separate CSV files.
- The system should merge these datasets based on the common identifier **Employee_ID**.
- The merged data should be processed to calculate the average salary per department.
- The system should output the processed data to an Excel file for easy sharing and reporting.
- The system must provide easy access to the first few rows and metadata of the merged dataset.

Constraints

Input Requirements

Employee Data:

- Must be stored as a CSV file in **employee_file**
- Should contain columns: **Employee_ID, Name, Department**
- Example column values: **101, John Doe, Marketing**

Salary Data:

- Must be stored as a CSV file in **salary_file**
- Should contain columns: **Employee_ID, Salary**
- Example column values: **101, 60000**

Data Merging Constraints

- The merging of employee and salary data must be done based on the **Employee_ID** column.
- Both CSV files must be available in the provided file paths.

Output Constraints

1. Display Format:

- The first 5 rows of the merged dataset should be displayed.
- Column names and data types for the merged DataFrame must be accessible.

2. Excel Output:

- The merged DataFrame should be saved as an Excel file named **merged_employee_data.xlsx**.

Functional Requirements

- **Data Loading**
The system should load the employee and salary CSV data into Pandas DataFrames.
- **Data Merging**
The system should merge the employee and salary data on the **Employee_ID** column.
- **Data Display**
The system should display the first 5 rows of the merged data for quick

inspection.

- **Metadata Retrieval**
The system should provide information about the DataFrame, including column names and data types.
 - **Salary Calculation**
The system should calculate the average salary per department by grouping the data by the **Department** column.
 - **Excel Export**
The system should save the merged data into an Excel file.
-

Template Code Structure:

1. Data Loading Functions:

- `load_employee_data()`
- `load_salary_data()`

2. Data Merging Function:

- `merge_data()`

3. Data Display Functions:

- `display_head()`
- `dataframe_info()`

4. Salary Calculation Function:

- `calculate_average_salary()`

5. Output Function:

- `save_to_excel()`

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) then you need to use **CTRL+Shift+B** -command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.
- 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 setup environment:
You can run the application without importing any packages
- To launch application:
python3 mainclass.py
- To run Test cases:
python3 -m unittest
- Before Final Submission also, you need to use **CTRL+Shift+B** - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.

Screen shot to run the program

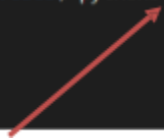
To run the application

```
OK
coder@dighe20250227t070305rz1fj5p3:/home/myproject/dighegmailcom_20250227T070305$ python3 <<scriptname>>.py
```



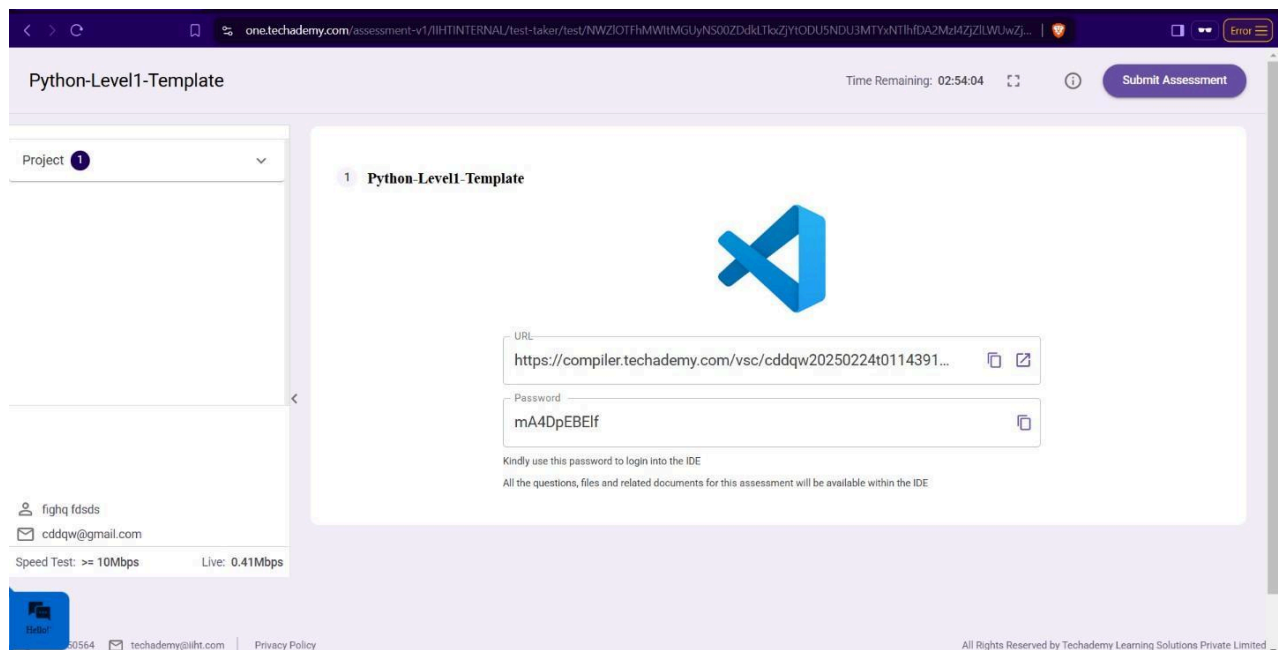
python3 mainclass.py

```
● coder@dighe20250227t070305rz1fj5p3:/home/myproject/dighegmailcom_20250227T070305$ python3 -m unittest
TestBoundary = Passed
.TestExceptional = Passed
.TestCalculateTotalDonations = Failed
.TestCalculateTotalStockValue = Failed
.TestCheckFrankWhiteDonated = Failed
```



To run the testcase

python3 -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.**