

System Requirements

Specification Index

For

Apply a Function to Transform All Values in a DataFrame Column Using the apply() Method

(Topic: Advanced Pandas)

Version 1.0

Employee Data Analysis Console

Project Abstract

The Employee Data Analysis Console is a Python-based tool developed to assist in analyzing and manipulating employee data stored in CSV format. This tool uses the Pandas library to load, process, and save employee-related data, focusing on increasing salaries, displaying summary information, and providing insights into the dataset. The application is essential for human resources departments to quickly adjust employee compensation and gather useful statistical information about employee data. This system streamlines the employee data analysis process and ensures that data handling and salary adjustments are performed accurately and efficiently.

Business Requirements:

- The system should be able to load employee data from a CSV file.
- The system should provide basic insights into the employee data, including the first five rows and column data types.
- The system should allow for increasing the salary of all employees by 10%.
- The system should allow saving the modified data into a new CSV file.

Constraints

Input Requirements:

- **Employee Data CSV:**
 - The file must contain employee data, including fields such as name, department, salary, and other relevant details.
 - The file format must be CSV.
 - The "Salary" column must be numeric.

Output Requirements:

- **Salary Adjustment:**
 - The salary of each employee will be increased by 10%.
 - The adjusted data should be saved into a new CSV file named "updated_employee_data.csv" or a custom name provided by the user.

Functional Constraints

1. CSV File Handling:

- The system must read the employee data from a provided CSV file.
- The system must save the updated employee data to a new CSV file.

2. Salary Adjustment Logic:

- The salary of each employee will be increased by 10% using the Pandas apply function.
- The updated salary should be reflected in the "Salary" column of the DataFrame.

3. Data Processing Constraints:

- Display the first 5 rows of the employee data in a readable format.
- Display DataFrame column names and data types.

Required Output Format:

• Salary Adjustment Output:

- After the adjustment, the system will save the updated employee data to a new CSV file.
- The new file will contain all the original columns, with the "Salary" column reflecting the increased values.

Template Code Structure:

1. Class and Methods:

- **EmployeeDataAnalysis:**
 - `__init__(self, file_path)` - Load employee data from a CSV file.
 - `display_head()` - Display the first 5 rows of the DataFrame.
 - `dataframe_info()` - Display DataFrame column names and data types.
 - `increase_salary()` - Increase the salary by 10%.
 - `save_to_csv()` - Save the updated DataFrame to a new CSV file.

2. Input Section:

- **Load employee data from a CSV file.**

3. Processing Section:

- **Display the first 5 rows of the dataset.**
- **Show column names and data types.**
- **Increase the salary by 10%.**

4. Output Section:

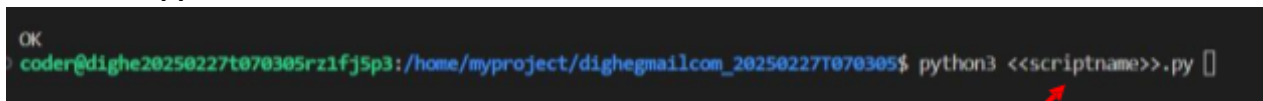
- **Save the updated data to a new CSV file.**

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

A screenshot of a terminal window with a dark background. The prompt is 'coder@dighe20250227t070305rz1fj5p3: /home/myproject/dighegmailcom_20250227T070305\$'. The command 'python3 <<scriptname>>.py' is entered. A red arrow points from the text 'python3 mainclass.py' below to the '<<scriptname>>.py' part of the command in the terminal.

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

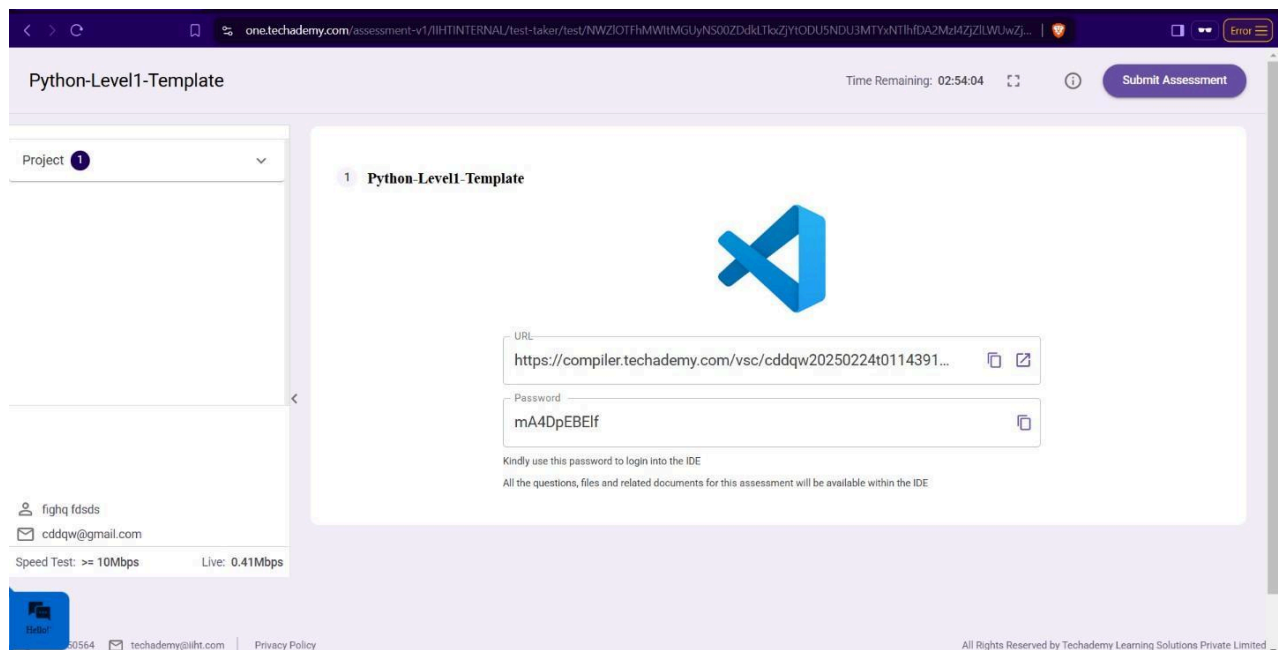
python3 mainclass.py

```
● 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.