

System Requirements

Specification Index

For

**Compare the Performance of Vectorized
Operations vs. Loops in Pandas Using timeit**

(Topic: Pandas Performance and Optimization)

Version 1.0

Employee Data Analysis System

Project Abstract

The Employee Data Analysis System is a Python-based application developed to analyze employee salary data stored in CSV format. This system processes the data to perform various operations, including displaying the first few rows, retrieving metadata about the dataset, and increasing employee salaries. The system also provides the ability to compare the performance of vectorized operations against traditional loop-based methods for salary adjustments. This application is essential for businesses that need to manage and analyze employee salary data efficiently and make data-driven decisions regarding compensation.

Business Requirements:

- Ability to load employee data from a CSV file or directly from a provided dataset.
 - Provide easy-to-understand data insights such as the first few rows and column metadata.
 - Implement salary increase functionality using both vectorized operations and loops.
 - Allow performance comparisons between different methods of salary increase.
 - Save the updated employee data back to a new CSV file.
-

Constraints

Input Requirements

- Employee Data:
 - Must be stored in a CSV file or provided directly as a dataset.
 - Must contain at least the columns: "Employee ID", "Name", and "Salary".

- Example: "Employee ID: 1, Name: 'John Doe', Salary: 50000"

Output Requirements

- **Updated Salary Data:**
 - Must save the updated dataset with salaries increased by 10% to a new CSV file.
 - Should format the output with proper column names and data.

Performance Constraints

- **Vectorized Operations:**
 - Must ensure that the salary increase operation is performed efficiently using vectorized methods.
- **Loop-Based Operations:**
 - Must provide a loop-based method for increasing salaries, but should highlight that this is less efficient than vectorized operations.

Conversion Constraints

- **Data Loading:**
 - If data is not provided, load data from a CSV file into a Pandas DataFrame.
- **Salary Increase (Vectorized):**
 - Increase salary by 10% for all employees using vectorized operations.
 - Must update the "Salary" column for all rows in the DataFrame.

- **Salary Increase (Loop):**
 - Increase salary by 10% using a loop.
 - Must iterate through the rows and modify each "Salary" value individually.
 - **Performance Comparison:**
 - Measure and compare the execution time of both vectorized and loop-based salary increase methods.
 - **Saving Updated Data:**
 - After performing the salary adjustments, save the updated DataFrame back to a CSV file.
-

Output Constraints

1. Display Format:

- Show the first 5 rows of the DataFrame when requested.
- Display column names and data types for the DataFrame.

2. Salary Adjustment:

- When using vectorized operations, ensure the salary column is updated for all employees.
- When using a loop, ensure the "Salary" values are updated in a similar manner but less efficiently.

3. Performance Metrics:

- Provide the time taken for both the vectorized operation and the loop-based method.

4. CSV File Saving:

- After the salary increase, save the updated dataset to a new CSV file named "updated_employee_data.csv".
-

Template Code Structure

1. Data Loading Functions:

- `load_data_from_csv()`
- `load_data_from_dataset()`

2. Salary Adjustment Functions:

- `increase_salary_vectorized()`
- `increase_salary_loop()`

3. Performance Measurement:

- `compare_performance()`

4. Output Functions:

- `display_head()`
- `display_data_info()`

5. File Saving Functions:

- `save_to_csv()`

6. Main Execution Logic:

- Load data from CSV or dataset.
- Perform salary increase using both methods.
- Compare performance and save the results.

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

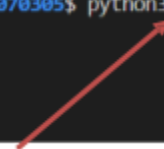
To run the application

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



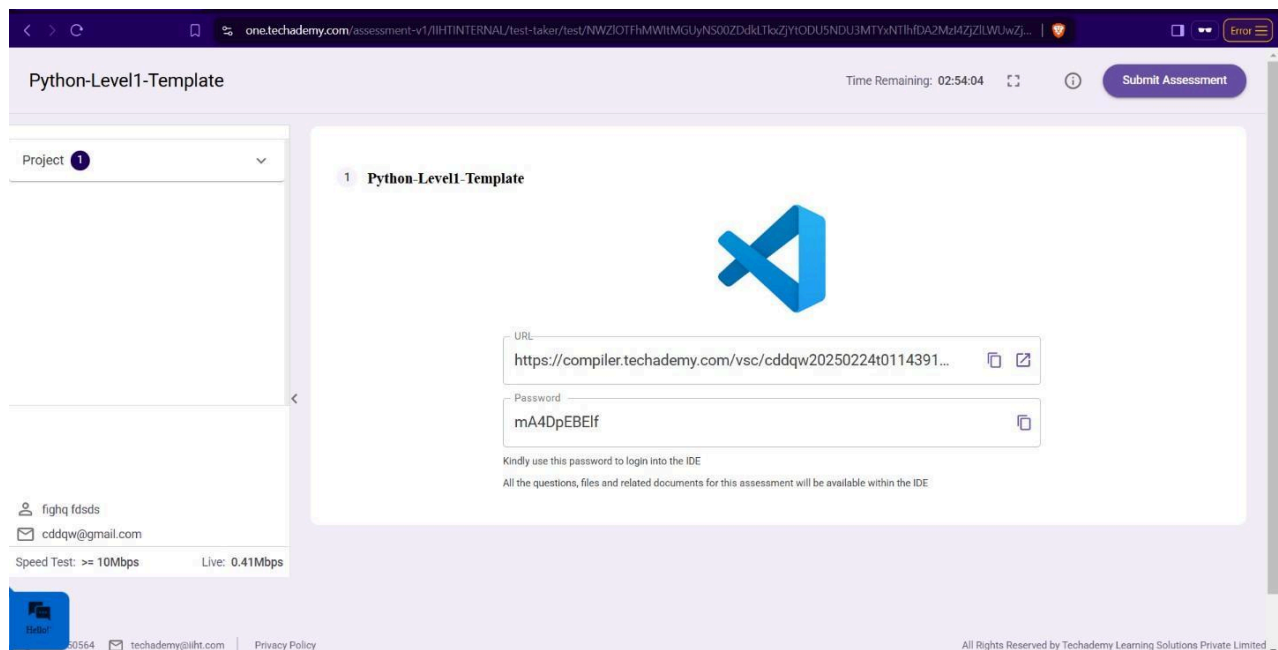
python3 mainclass.py

```
● 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.**