 

LADOKE AKINTOLA UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND INFORMATICS

DEPARTMENT OF COMPUTER SCIENCE

COURSE CODE: CSC 202

COURSE TITLE: COMPUTER PROGRAMMING II

PROJ ECT OVERVIEW: EMPLOYEE PERFORMANCE ANALYZER AND HR CALCULATOR

Create a human resources analysis tool that processes employee performance data and attendance records, calculates productivity metrics and performance ratings, implements salary analysis and compensation benchmarking, analyzes employee turnover patterns and retention factors, performs workload distribution and team efficiency calculations, and generates HR reports with performance improvement recommendations and workforce planning insights.

BY: GROUP 54

 

 

GROUP 54 MEMBERS



|  |  |
| --- | --- |
| Nafiu Wariz Olajide | 202301 0661 |

|  |  |
| --- | --- |
| Alamu Oluwadamilare Akintunde | 202300481 6 |

Muftau Fatimoh Morenikeji 2023007449

|  |  |
| --- | --- |
| Oyekola Musharaf Olamide | 202301 01 84 |

Hammed Faruq Olatunbosun 2023004999

 

 

This is an Employee Performance Analyzer & HR Calculator

Author : Csc'28 (Group 54)   
Python : 3.8+   
Description:   
This is a Command-Line Interface (CLI) application for managing and analyzing employee performance and HR metrics.It supports:

- Adding staff or manager records  
- Viewing summary statistics  
- Listing top performers  
- Saving/loading employee data  
- Exiting the system

T o run:   
 python employee\_app.py

import json #Saves and loads employee data   
import logging

|  |  |
| --- | --- |
| from dataclasses import dataclass, asdict #Defines data containers for employees |  |

 

from datetime import date #allows date input to be read from pathlib import Path #Used for file path operations from typing import Dict, List, Any   
from statistics import mean #Calculates Statistical data

logging.basicConfig(level=logging.INFO)

DATA\_FILE = Path("employee\_data.json")

class DataValidationError(Exception):   
 pass

# Object oriented programming(Each employee is an object) @ dataclass   
class Employee:   
 emp\_id: str   
 name: str   
 role: str   
 salary: float   
 department: str   
 hire\_date: str   
 tasks\_completed: int   
 tasks\_assigned: int   
 attendance\_days: int   
 working\_days: int

# Calculates the employee's productivity, attendance rates and rates based on his/her....

def productivity(self) -> float:   
 return (self.tasks\_completed / self.tasks\_assigned \* 1 00) if self.tasks\_assigned else 0.0

def attendance\_rate(self) -> float:   
 return (self.attendance\_days/ self.working\_days \* 1 00) if self.working\_days else 0.0

def rating(self) ->str:   
 score = (self.productivity() \* 0.7) + (self.attendance\_rate() \* 0.3) if score >= 90:   
 return "A"   
 elif score >= 75:   
 return "B"   
 elif score >= 60:   
 return "C"   
 return "D"

# This section fetches & manipulates employee data   
class HRSystem:   
 def \_\_init\_\_(self):   
 self.employees: Dict[str, Employee] = {}

 

 

def add\_employee(self, data: Dict[str, Any]):   
 if data["emp\_id"] in self.employees:   
 raise DataValidationError("Employee ID already exists. ") # Errorcatching for when you enter the same id for two people

self.employees[data["emp\_id"]] = Employee(\*\*data)

def save(self):   
 with open(DATA\_FILE, "w") as f:   
 json.dump([asdict(emp) for emp in self.employees.values()], f, indent=2)

def load(self):   
 if not DATA\_FILE.exists():   
 return   
 with open(DATA\_FILE, "r") as f:   
 raw = json.load(f)   
 for itemin raw:   
 self.employees[item["emp\_id"]] = Employee(\*\*item)

def average\_salary(self) -> float:   
 if not self.employees:   
 return 0.0   
 return mean(emp.salary for emp in self.employees.values())

def top\_performers(self, n=3) -> List[Employee]:   
 return sorted(self.employees.values(), key=lambda e: (e.rating(), e.productivity()), reverse=True)[:n]

# This is where we enter data related to employees def prompt\_employee\_data() -> Dict[str, Any]:   
 print("Enter Employee Info:")   
 try:   
 emp\_id = input("ID: ").strip()   
 name = input("Name: ").strip()   
 role = input("Role: ").strip()   
 salary = float(input("Salary: "))   
 department = input("Department: ").strip()   
 hire\_date = input("Hire Date (YYYY-MM-DD): ").strip() tasks\_completed = int(input("Tasks Completed: ")) tasks\_assigned = int(input("Tasks Assigned: "))   
 attendance\_days = int(input("Attendance Days: ")) working\_days = int(input("Working Days: "))

date.fromisoformat(hire\_date) # validates date

return {   
 "emp\_id": emp\_id,

|  |  |  |
| --- | --- | --- |
|  | "name": name, |  |

 

"role": role,   
 "salary": salary,   
 "department": department,   
 "hire\_date": hire\_date,   
 "tasks\_completed": tasks\_completed,   
 "tasks\_assigned": tasks\_assigned,   
 "attendance\_days": attendance\_days,   
 "working\_days": working\_days,   
 }   
 except Exception as e:   
 raise DataValidationError(f"Invalid input: {e}") #catches errors in input of employee data def main():   
 hr = HRSystem()   
 hr.load()

# Code that displays your options/menu (Interface)   
 while True:   
 print("\n--- HR CLI ---")   
 print("1 .Add Employee")   
 print("2.View Average Salary")   
 print("3.Show T op Performers")   
 print("4.Save & Exit")   
 choice = input("Choose an option: ").strip()   
 # Input response for the Interface   
 try:   
 if choice == "1 ":   
 data = prompt\_employee\_data()   
 hr.add\_employee(data)   
 print("Employee added. ")   
 elif choice == "2":   
 print(f"Average Salary: {hr.average\_salary():.2f}")   
 elif choice == "3":   
 top = hr.top\_performers()   
 for emp in top:   
 print(f"{emp.name}({emp.emp\_id}) - Rating: {emp.rating()}") elif choice == "4":   
 hr.save()   
 print("Data saved.Exiting. ")   
 break   
 else:   
 print("Invalid option. ")   
 except DataValidationError as ve:   
 print(f"Error: {ve}")   
 except Exception as e:   
 print(f"Unexpected error: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

|  |  |  |
| --- | --- | --- |
|  | main() |  |