```
/*TASK [3]:
Employee and attrition and performance:
In this project, you will need to evaluate and its relationship
with attrition, for each factor X
1.example, the distance from home to office, the job role impact
on attrition, etc.*/

/*The algorithm for this code to understand and use it:
1} Define the Employee structure with attributes:
distanceFromHome, jobRoleImpact, performance, and attrition.
2} Implement the trainLogisticRegression function:
    -Initialize weights and bias to zero.
    -Iterate for a specified number of iterations:
    -Calculate the prediction using the logistic regression
formula.
    -Print the iteration number and the mean error.
    -If the mean error is below a threshold, exit the loop.
3} Implement the predictAttrition function:
    -Calculate the prediction using the logistic regression
formula.
    -Return true if the prediction is greater than or equal to
0.5, indicating attrition, and false otherwise.
4} In the main function:
    -Create a vector of Employee objects with some sample data.
    -Create an empty map of weights and initialize the bias.
    -Set the number of iterations and the learning rate.
    -Call the predictAttrition function with the trained weights,
bias, and the entered employee data.
    -Print the predicted attrition value.*/


/*CODE IN C++ FORMAT*/
#include <iostream>
#include <vector>
#include <map>
#include <cmath>
using namespace std;

// Structure employee
struct Employee
{
    double distanceFromHome;
    double jobRoleImpact;
    double performance;
    bool attrition;
};

// Function to train a logistic regression model
void trainLogisticRegression(const vector<Employee>& employees,
map<string, double>& weights, double& bias, int numIterations,
double learningRate)
{
```

```
    // Initialize weights and bias
    weights["distanceFromHome"] = 0.0;
    weights["jobRoleImpact"] = 0.0;
    weights["performance"] = 0.0;
    bias = 0.0;
    for (int iteration = 0; iteration < numIterations;
iteration++)
    {
        double errorSum = 0.0;
        double biasGradient = 0.0;

        for (const Employee& employee : employees)
        {
            double prediction = 1.0 / (1.0 + exp(-1.0 *
(weights["distanceFromHome"] * employee.distanceFromHome +

weights["jobRoleImpact"] * employee.jobRoleImpact +

weights["performance"] * employee.performance + bias)));
            double error = employee.attrition - prediction;

            weights["distanceFromHome"] += learningRate * error *
employee.distanceFromHome;
            weights["jobRoleImpact"] += learningRate * error *
employee.jobRoleImpact;
            weights["performance"] += learningRate * error *
employee.performance;
            bias += learningRate * error;

            errorSum += error;
            biasGradient += error;
        }

        double meanError = errorSum / employees.size();
        double meanBiasGradient = biasGradient /
employees.size();

        cout << "Iteration: " << iteration + 1 << ", Mean Error:
" << meanError << endl;

        // Stop training if the mean error is below a certain
threshold
        if (fabs(meanError) < 1) {
            break;
        }
    }
}

// Function to predict attrition based on employee factors
bool predictAttrition(const map<string, double>& weights, double
bias, double distanceFromHome, double jobRoleImpact, double
performance)
```

```
{
    double prediction = 1.0 / (1.0 + exp(-1.0 *
(weights.at("distanceFromHome") * distanceFromHome +

weights.at("jobRoleImpact") * jobRoleImpact +

weights.at("performance") * performance + bias)));
    return prediction >= 0.5;
}

int main()
{
    // Employee data
    std::vector<Employee> employees =
    {
        { 5.0, 0.8, 4.5, false },
        { 10.0, 0.3, 3.2, true },
        { 3.0, 0.9, 4.7, false },
        // You can add more data to justify  for better
predection
    };

    // Training the logistic regression model
    map<string, double> weights;
    double bias;
    int numIterations = 10;
    double learningRate = 01;
    trainLogisticRegression(employees, weights, bias,
numIterations, learningRate);

    // Prediction attrition for a new employee are joining
    double distanceFromHome;
    double jobRoleImpact;
    double performance;
    cout << "Enter the distance from home: ";
    cin >> distanceFromHome;
    cout << "Enter the job role impact: ";
    cin >> jobRoleImpact;
    cout << "Enter the performance: ";
    cin >> performance;

    bool attrition = predictAttrition(weights, bias,
distanceFromHome, jobRoleImpact, performance);
    cout << "Predicted attrition: " << (attrition ? "Yes" : "No")
<< endl;

    return 0;
}
//END OF CODE {Thank you}
```