**System Requirements Specification Index**

**For Machine learning Algorithm No 4**

1.0

**Machine Learning Assessment: Logistic regression and Linear Regression**

**Machine Learning Assessment Instructions**

**Overview**

**This assessment consists of two Python files with skeleton code that you need to implement:**

**1. `employee.py` - Employee attrition prediction using logistic regression**

**2. `student.py` - Student score prediction using linear regression**

**Dataset Information**

**Employee Attrition Dataset (`employee\_attrition.csv`)**

This dataset contains information about employees and whether they left the company (attrition).

Key columns:

- `satisfaction\_level`: Employee satisfaction level (0-1)

- `last\_evaluation`: Score of last performance evaluation (0-1)

- `number\_project`: Number of projects the employee is involved in

- `average\_monthly\_hours`: Average monthly working hours

- `time\_spend\_company`: Years at the company

- `work\_accident`: Whether the employee had a workplace accident (0/1)

- `left`: Whether the employee left the company (0/1) - This is the target variable

- `promotion\_last\_5years`: Whether the employee was promoted in the last 5 years (0/1)

**-** `department`: Department the employee works in

- `salary\_level`: Salary level (low/medium/high)

**Student Scores Dataset (`student\_scores.csv`)**

**This dataset contains information about students and their final exam scores.**

Key columns:

- `hours\_studied`: Number of hours studied

- `previous\_score`: Score in the previous exam

- `assignments\_completed`: Number of assignments completed

- `final\_score`: Final exam score - This is the target variable

**Task Overview**

Your task is to implement the functions in both files according to the TODO comments. Each function has detailed instructions on what it should do. The skeleton code is designed to fail the tests initially, and your implementation should make the tests pass.

Detailed Implementation Instructions

Employee Attrition Prediction (`employee.py`)

**Function 1: `load\_and\_prepare\_data(path="employee\_attrition.csv")`**

**Purpose: Load and preprocess the employee attrition dataset.**

Implementation Steps:

1. Use `pd.read\_csv(path)` to load the dataset

2. Print statistics about average monthly hours using python

   print("\n📊 Avg Monthly Hours - Mean: {:.2f}, Max: {:.2f}".format(

       df['average\_monthly\_hours'].mean(), df['average\_monthly\_hours'].max()))

   3. For categorical columns ('department', 'salary\_level'), check if they are object type and encode them using `LabelEncoder()`

4. Scale all features (except 'left') using `StandardScaler()`

5. Print a success message: `print(" Dataset loaded and preprocessed.")`

6. Return the prepared DataFrame

**Function 2: `hypothesis\_demo()`**

**Purpose: Demonstrate the logistic regression hypothesis function.**

**Implementation Steps:**

1. Create a sample feature vector, e.g., `x\_sample = np.array([0.5, -1.2, 0.8])`

2. Define weights, e.g., `weights = np.array([1.5, -0.8, 2.0])`

3. Define bias, e.g., `bias = 0.3`

4. Calculate z = dot product of weights and x\_sample plus bias

5. Calculate h(x) = sigmoid(z) = 1 / (1 + np.exp(-z))

6. Print information about the hypothesis

   print(f"\n Hypothesis h(x) = sigmoid(w·x + b)")

   print(f" z = {z:.4f}")

   print(f" Probability that employee will leave = {h\_x:.4f}")

**Function 3: `sigmoid\_demo()`**

Purpose: Demonstrate the sigmoid activation function.

Implementation Steps:

1. Define z = 2.0

2. Calculate sigmoid(z) = 1 / (1 + np.exp(-z))

3. Print the result: `print(f"\n Sigmoid(2.0) = {sigmoid:.4f}")`

   - Note: The result should be 0.8808

**Function 4: `cost\_function(y\_true, y\_pred\_prob)`**

Purpose: Implement the log loss cost function.

Implementation Steps:

1. Add a small epsilon (e.g., 1e-15) to prevent log(0)

2. Clip prediction probabilities using `np.clip(y\_pred\_prob, epsilon, 1 - epsilon)`

3. Calculate binary cross-entropy:-np.mean(y\_true \* np.log(y\_pred\_prob) + (1 - y\_true) \* np.log(1 - y\_pred\_prob))

  4. Return the calculated cost

**Function 5: `train\_and\_evaluate(X\_train, y\_train, X\_test, y\_test, path="attrition\_model.pkl")`**

Purpose: Train and evaluate a logistic regression model.

Implementation Steps:

1. Create a LogisticRegression model with max\_iter=1000

2. Train the model on X\_train and y\_train

3. Save the model to the specified path using `joblib.dump(model, path)`

4. Print a success message: print(f"\n Model trained and saved to '{path}'")`

5. Make predictions on X\_test (both class predictions and probabilities)

6. Calculate the cost using your custom cost\_function

7. Print the cost and sample predictions

    print(f"\n Log Loss (Custom Cost): {cost:.4f}")

   print(“Sample Predictions:", y\_pred[:10])

**second Test file Student Score Prediction (`student.py`)**

**Function 1: `load\_and\_preprocess(path)`**

Purpose: Load and preprocess the student scores dataset.

Implementation Steps:

1. Use `pd.read\_csv(path)` to load the dataset

2. Convert column names to lowercase and strip whitespace: `df.columns = df.columns.str.lower().str.strip()`

3. Remove rows with missing values using `df.dropna()`

4. Print a success message: `print("📚 Student data loaded and cleaned.")`

5. Return the cleaned DataFrame

**Function 2: `show\_key\_stats(df)`**

**Purpose: Display key statistics about the dataset.**

Implementation Steps:

1. Calculate the standard deviation of hours\_studied: `hours\_std = df['hours\_studied'].std()`

2. Find the maximum value of previous\_score: `max\_previous\_score = df['previous\_score'].max()`

3. Print these statistics:

   print(f"\n Standard Deviation of Study Hours: {hours\_std:.2f}")

   print(f" Max Previous Score: {max\_previous\_score}")

**Function 3: `prepare\_data(df, features, target)`**

Purpose: Prepare the data for model training.

Implementation Steps:

1. Extract features (X) and target (y) from the DataFrame

2. Create a StandardScaler and scale the features

3. Split the data into training and testing sets (80/20 split) with random\_state=42

4. Print a success message: `print("\n Data prepared and split.")`

5. Return X\_train, X\_test, y\_train, y\_test, and the scaler

**Function 4: `train\_and\_save\_model(X\_train, y\_train, model\_path="student\_score\_model.pkl")`**

**Purpose: Train and save a linear regression model.**

Implementation Steps\*\*:

1. Create a LinearRegression model

2. Train the model on X\_train and y\_train

3. Save the model to the specified path using `joblib.dump(model, model\_path)`

4. Print a success message: `print(f"\n Model trained and saved to '{model\_path}'")`

5. Return the trained model

Function 5: `evaluate\_model(model, X\_test, y\_test)`

**Purpose: Evaluate the model performance.**

**Implementation Steps:**

1. Make predictions on X\_test

2. Calculate the mean squared error using `mean\_squared\_error(y\_test, y\_pred)`

3. Print the MSE and sample predictions:

   print(f"\n Mean Squared Error: {mse:.2f}")

   print(" Sample Predictions:", y\_pred[:5])

**Implementation Guidelines**

**1. Read the TODO comments carefully to understand what each function should do**

**2. Implement each function according to the specifications**

**3. Make sure your implementation passes all the test cases in `test/test\_functional.py`**

**4. Do not modify the function signatures or return types**

**python -m test.test\_functional**

**If all tests pass, your implementation is correct. If any tests fail, review the error messages and fix your implementation accordingly.**

**Common Pitfalls to Avoid**

1. Not printing the exact expected messages: Make sure your print statements match exactly what's expected in the tests.

2. Incorrect function signatures : Don't change the function parameters or return types.

3. Not handling edge cases : Make sure your functions handle potential errors gracefully.

4. Incorrect scaling or encoding: Follow the instructions carefully for preprocessing steps.

5. Not saving models correctly : Make sure you're using joblib.dump correctly to save models.

**Additional Notes**

**- The `employee.py` file implements a logistic regression model for binary classification**

**- The `student.py` file implements a linear regression model for predicting continuous values**

**- Both files include a main section that demonstrates the full workflow**

**- Make sure to print the required messages as specified in the TODO comments**

**Important Notes**

**1. Make sure to import all necessary libraries at the beginning of each file.**

**2. Follow the function signatures exactly as specified in the skeleton code.**

**3. Ensure that your functions print the expected output messages.**

**4. The random\_state parameter should be set to 42 for reproducibility.**

**5. Pay attention to the return values of each function, as they are used in subsequent tests.**

**Running the Tests**

To run the tests, use the following command:

**Python3 -m unittest**

**Submission Guidelines**

1. Complete all the required functions in `Employee.py` and `Student.py`

2. Ensure all tests pass

3. Submit your code files

**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 employee .py**

**Python3 Student.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.

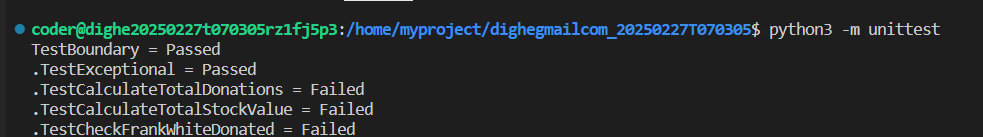
**Screen shot to run the program**

**To run the application**

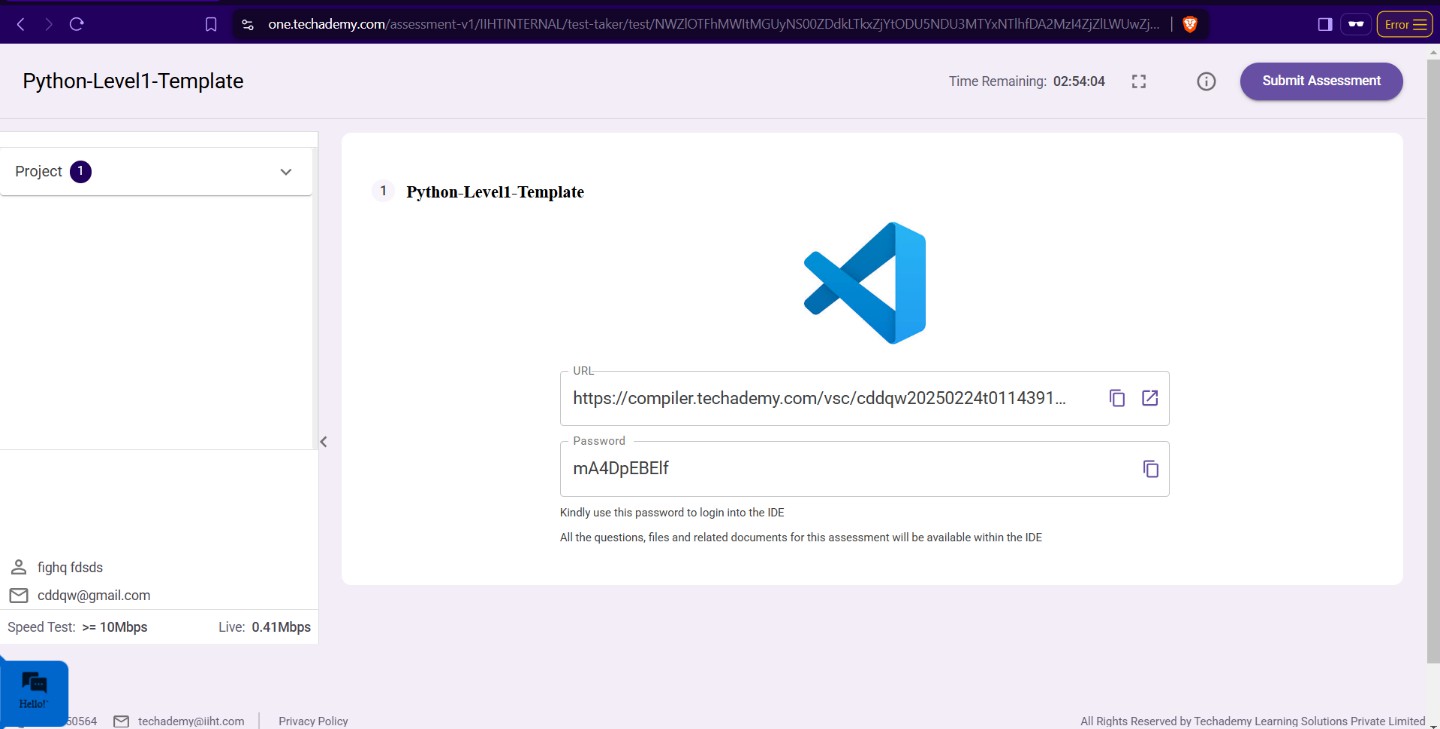


**Python3 employee .py**

**Python3 Student.py**



**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.**