**System Requirements Specification Index**

**For Machine Learning Algorithm No. 6**

1.0

**Machine Learning use case: Decision Trees and Random Forests**

**Heart Disease and Chicken Disease Prediction**

**Overview**

**This assessment focuses on implementing two machine-learning models for disease prediction:**

1. Heart Disease Prediction using a Random Forest Classifier

2. Chicken Disease Prediction using a Decision Tree Classifier

You will implement code that provides functionality for data loading, preprocessing, model training, and prediction. The goal is to understand the core concepts of data preprocessing, model training, and evaluation for classification tasks.

**Datasets information**

1. Heart Disease Dataset (`heart.csv`)

This dataset contains medical information for 303 patients, with a target variable indicating the presence of heart disease.

Features:

- `age`: Age in years

- `sex`: Sex (1 = male, 0 = female)

- `cp`: Chest pain type (0-3)

- `trestbps`: Resting blood pressure (in mm Hg)

- `chol`: Serum cholesterol in mg/dl

- `fbs`: Fasting blood sugar > 120 mg/dl (1 = true, 0 = false)

- `restecg`: Resting electrocardiographic results (0-2)

- `thalach`: Maximum heart rate achieved

- `exang`: Exercise induced angina (1 = yes, 0 = no)

- `oldpeak`: ST depression induced by exercise relative to rest

- `slope`: Slope of the peak exercise ST segment

- `ca`: Number of major vessels colored by fluoroscopy (0-3)

- `thal`: Thalassemia (3 = normal, 6 = fixed defect, 7 = reversible defect)

Target:

- `target`: Presence of heart disease (1 = yes, 0 = no)

2. Chicken Disease Dataset (`chicken\_disease\_data.csv`)

This dataset contains information about 1000 chickens, with features related to their health and a target variable indicating disease status.

Features:

- `Age`: Age of the chicken in years

- `Weight`: Weight of the chicken in kg

- `Height`: Height of the chicken in cm

- `Egg Production`: Number of eggs produced per week

- `Feed Consumption`: Amount of feed consumed in kg per week

- `Temperature`: Body temperature in Celsius

- `Disease Predicted`: Disease status (Healthy or Sick)

**Tasks**

**Task 1: Heart Disease Prediction**

You will implement the following functions in `heart.py`:

1. `load\_heart\_disease\_data()`: Load the heart disease dataset

* Load the CSV file named heart.csv using pandas.
* The user can use the print statement to check if the dataset has only 303 records
* Return the dataset as a pandas.DataFrame

Expected columns {

"age", "sex", "cp", "trestbps", "chol", "fbs",

"restecg", "thalach", "exang", "oldpeak",

"slope", "ca", "thal", "target"

}

2. **preprocess\_heart\_data(df):Separate features (X) and target (y)**

* In the next step you are requested to accept the DataFrame df as input.
* X should contain all columns except "target"
* y should contain only the "target" column
* Drop the column "target" from df and store the remaining columns in X (features)
* Extract the "target" column from df and store it in y (label).
* Return X and y

3. **split\_heart\_data(X, y, test\_size=0.2): Split features and labels into training and testing sets**

In this function, the user is required to split the dataset

* Use train\_test\_split from sklearn.model\_selection to split the input features X and target y.
* Use the given test\_size parameter to determine the size of the test set (default is 20%).
* Set random\_state=42 to ensure reproducible splits.
* Return the split data as: X\_train, X\_test, y\_train, y\_test.
* X\_train: features for training the model
* X\_test: features for evaluating the model
* y\_train: Target labels for training
* y\_test: Target labels for testing

4. **create\_train\_save\_load\_model (X\_train, y\_train, n\_estimators=100, max\_depth=None,  
 filename="random\_forest\_heart\_model.pkl"):**

**Create → Train → Save → Load → Return a model**

In the function, the user is required to create the model. The user first has to

* Initialize a RandomForestClassifier from sklearn inbuilt library
* The parameters the user is required to use are (n\_estimators ,max\_depth )
* Use the n\_estimators parameter to specify the number of trees in the forest (default is 100).
* Use max\_depth to control the maximum depth of each tree (default is None, meaning nodes are expanded until all leaves are pure).
* Set random\_state=42 to ensure reproducibility.
* When the user calls the function while passing the value the user is required to use max\_depth =max\_depth
* **Train**: Use the fit() method to train the model using the training data X\_train and y\_train.
* **Save**: After training, save the trained model using joblib.dump() to a file specified by the filename parameter (default is "random\_forest\_heart\_model.pkl").
* **Load**: After saving, immediately load the model back from the file using joblib.load().
* **Return**: Return the loaded model for further use (e.g., making predictions).

**6 . Predict heart disease for new data**

**check\_new\_data\_from\_json(model, json\_file="heart\_data.json"): This** function is used to predict whether a new patient has heart disease or not using a trained model.

* **Input**:
  + Model: The trained model that will be used to make predictions.
  + Use the json\_file with the path to a JSON file containing patient data (heart\_data.json).
* **Process**:
  + Extract the patient's data and convert it to a pandas.DataFrame.
  + Use the model to predict whether the patient has heart disease (output 1 for "YES", 0 for "NO").
* **Return**: Return the prediction value (1 for heart disease, 0 for no heart disease).

**Task 2: Chicken Disease Prediction**

You will implement the following functions in `chicken.py`:

1. Complete the function named load\_chicken\_disease\_data().
2. Load the dataset from a CSV file named "chicken\_disease\_data.csv" using the pandas.read\_csv() function.
3. Limit the DataFrame to the **first 1000 rows** using appropriate slicing or the .head() method.
4. Return the dataset as a pandas.DataFrame

Expected columns  {

                "Age", "Breed", "Temperature”, "Eating Behavior", "Coughing",  "Feces Appearance",

                "Water Consumption", "Disease Predicted”}

2. `perform\_eda\_on\_age(df)`: Perform exploratory data analysis on the Age column

* The user needs to check the **"Age"** column of the DataFrame df and perform a simple exploration to count how many records are in chicken\_disease\_data.csv with the condition.
* You should write the code to check if the Age column exists
* If the **"Age"** column exists, it filters the dataset to find all rows where the **"Age"** is greater than **2**
* **You can use the .shape[0] method to check how many records meet this condition**
* Return count

3. `preprocess\_chicken\_data(df): Preprocess the chicken disease data

In this function,

* The code should be written in the function to confirm that the dataset contains a column named "**Disease Predicted**"
* Map the dataset to Healthy = 0, Sick = 1 and check for unmapped valued if any .
* Apply one-hot encoding (pd.get\_dummies()) to all categorical features.
* Use drop\_first=True to avoid redundant dummy variables.
* Split the processed DataFrame into:
* X → all features (drop the target)
* y → the target column
* Return a tuple: (X, y, processed\_df)

4. `split\_chicken\_data(X, y, test\_size=0.2)`: Split the data into training and testing sets

In this function the user is required to split the dataset

* Use train\_test\_split from sklearn.model\_selection to split the input features X and target y.
* Use the given test\_size parameter to determine the size of the test set (default is 20%).
* Set random\_state=42 to ensure reproducible splits.
* Return the split data as: X\_train, X\_test, y\_train, y\_test.
* X\_train: features for training the model
* X\_test: features for evaluating the model
* y\_train: Target labels for training
* y\_test: Target labels for testing

5. create\_and\_train\_model(X\_train, y\_train)`: Create and train a Decision Tree model

* The function first creates an instance of the DecisionTreeClassifier from the sklearn.tree library. The random\_state=42 ensures that the model training is reproducible.
* **Train**: Use the fit() method to train the model using the training data X\_train and y\_train.
* **Save**: After training, save the trained model using joblib.dump() to a file specified by the filename parameter (default is "decision\_tree\_chicken\_disease\_model.pkl").
* **Load**: After saving, immediately load the model back from the file using joblib.load().
* **Return**: Return the loaded model for further use (e.g., making predictions).

6. check\_new\_data\_from\_json(model, json\_file="chicken\_data.json"):

This function is used to predict whether a new chicken has a disease or not using a trained model.

* **Input**:
  + Model: The trained model that will be used to make predictions.
  + Use the json\_file with the path to a JSON file containing patient data (chicken\_data.json).
* **Process**:
  + Extract the chicken data and convert it to a pandas.DataFrame.
  + Use the model to predict whether the chicken has the disease (output 1 for "YES", 0 for "NO").

**Return Value:**

* The function returns the final prediction:
  + 1 if the chicken is predicted to have a disease (sick).
  + 0 if the chicken is predicted to be healthy

To run the tests, use the following command:

Python3 -m unittest

**Submission Guidelines**

1. Complete all the required functions in `heart.py` and `chicken.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 heart.py` and `python3 chicken.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.

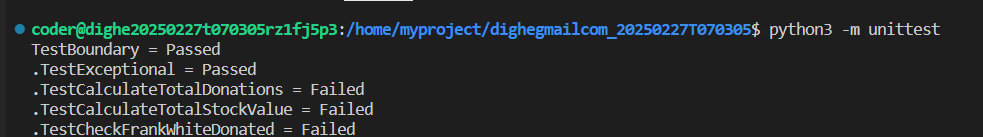
**Screen shot to run the program**

**To run the application**

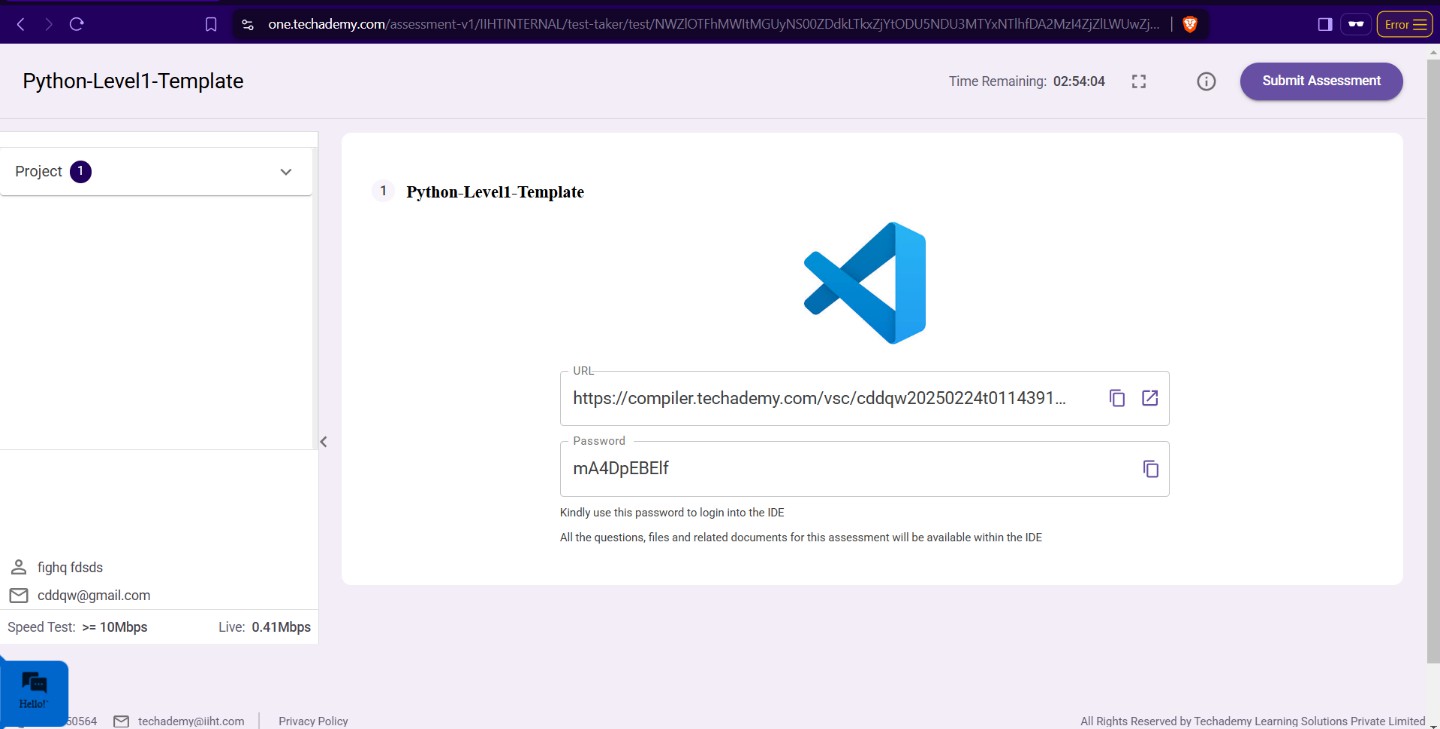


**Python3 heart.py**

**python3 chicken.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.**