# System Requirements Specification Index

## For

## Pytorch Mental illness use case L2

**IIHT Pvt. Ltd.**

fullstack@iiht.com

## Mental Health Prediction using PyTorch

## Objective:

In this project, we uses machine learning to classify employees into wellbeing categories—**low**, **medium**, or **high**—based on this data. The model can help organizations take proactive steps, such as offering support or recognizing high performers, to promote a healthier workplace.

**Dataset information**

## Dataset Column Names and Descriptions

| Column Name | Meaning / Description |
|---|---|
| `sleep_hours` | Average number of hours the individual sleeps per night. Adequate sleep is generally associated with better mental health. |
| `screen_time_hours` | Total number of hours spent daily on screens (e.g., phones, computers, TV). High screen time may correlate with stress or low productivity. |
| `exercise_minutes` | Number of minutes spent exercising per day. Physical activity can positively impact mental wellbeing. |
| `stress_level` | Self-reported stress level on a scale from **1 (low)** to **5 (high)**. Higher stress levels negatively influence wellbeing. |
| `productivity_score` | Self-assessed productivity on a scale from **0 to 100**, where a higher value indicates better perceived productivity. Used as a base for calculating the wellbeing score. |

The score is then **binned** into three classes:

| Class | Condition | Meaning |
|---|---|---|
| 0 | wellbeing_score ≤ 40 | Low Wellbeing |
| 1 | 40 < wellbeing_score ≤ 70 | Medium Wellbeing |
| 2 | wellbeing_score > 70 | High Wellbeing |

**The information which are provided to you are**

- One CSV file with the file name **mental_health_data.csv**
- One text file for introducing new data to detect, with the file name **new_user_input.txt**
- **Main.py is provided with template code where you need to implement the code**
- **You need to save the model with same name of mental_model_class.pth**
- **Accuracy of your model create should be above 80 percent**

---

## 1. `bin_wellbeing` – Create a Custom Labeling Function

- Define a function `bin_wellbeing(productivity, stress)` to compute a "wellbeing score" using the formula: `score = productivity - (stress * 5)`.
- Use `pandas.cut()` to categorize the score into three classes:
  - `Class 0` for scores ≤ 40,
  - `Class 1` for scores > 40 and ≤ 70,
  - `Class 2` for scores > 70.
- Return the labels as an integer series using `.astype(int)`.

---

## 2. `load_data_from_csv` – Data Preprocessing Function

- Define a function to load data from a CSV file (default: `'mental_health_data.csv'`).
- Use `pandas.read_csv()` to read the file.
- Generate the target label by passing `productivity_score` and `stress_level` columns to the `bin_wellbeing` function.
- Drop any rows with null values in the generated labels.
- Drop the `productivity_score` column from the DataFrame to prepare features `X`.
- Scale the features using `StandardScaler`.
- Use `StratifiedShuffleSplit` from `sklearn.model_selection` to split the dataset into training and testing sets.
- Convert the split arrays into PyTorch tensors (float32 for features, long for labels).
- Return the training and testing tensors along with the fitted scaler.

---

## 3. `MentalHealthDataset` – Create PyTorch Dataset Class

- Create a subclass of `torch.utils.data.Dataset` named `MentalHealthDataset`.
- Implement the constructor `__init__` to accept and store tensors `X` and `y`.
- Define the `__len__` method to return the number of samples in the dataset using `len(self.X)`.
- Define the `__getitem__` method to return a tuple `(X[idx], y[idx])` for a given index `idx`.

---

### 4. `build_model`

- Create a function to build a PyTorch `nn.Sequential` model with the following layers:
  - `Linear(input_size, 16)`, followed by `ReLU()`, and `Dropout(0.3)`
  - `Linear(16, 8)`, followed by `ReLU()`
  - `Linear(8, num_classes)`
- Return the constructed model.

---

## 5. `train_model` – Implement the Training Loop

- Define a function to train the model using:
  - `CrossEntropyLoss` as the loss criterion,
  - `Adam` optimizer.
- Loop through the given number of `epochs`.
- For each batch in the dataloader:
  - Zero out gradients.
  - Perform a forward pass to get outputs.
  - Calculate loss and perform backward pass.
  - Update model weights with the optimizer.
- If a validation dataloader is provided, compute and print the accuracy at the end of each epoch.(→ count should be 15 )

---

## 6. `evaluate_model` – Model Evaluation Logic

- Define a function that sets the model to evaluation mode using `model.eval()`.
- Use `torch.no_grad()` to prevent gradient tracking during inference.
- Loop over the evaluation data, get predictions using `torch.max(outputs, 1)`, and compare them to actual labels.
- Calculate and return the overall accuracy as `correct / total`.

---

## 7. `load_new_user_data` – Load Inference Input from File

- Define a function to read a single-line text file (default: `'new_user_input.txt'`) containing comma-separated values like `"7.0,4.5,30,2"`.
- Read the line, split by comma, and convert each value to a float.
- Return a NumPy array of shape `(1, n_features)`.

---

## 8. `predict_new_user` – Predict Class for New User Input

- Create a function to:
  - Rebuild the model using `build_model`,
  - Load the saved model weights using `load_state_dict`,
  - Load the user input from the file using `load_new_user_data`,
  - Scale the input using the `scaler.transform()`,
  - Convert input to a PyTorch tensor and predict using `model()`,
  - Return the predicted class using `torch.argmax()`.

---

## 9. `save_model` – Save the Trained Model

- Define a function that takes the model and saves its state dictionary using:

- **Follow these steps inside:**
    1. Load and preprocess the dataset using `load_data_from_csv`.
    2. Create training and testing `MentalHealthDataset` objects.
    3. Use `DataLoader` with batch size of 4 to create `train_loader` and `test_loader`.
    4. Build the model using `build_model`.
    5. Train the model for 15 epochs using `train_model`.
    6. Evaluate the trained model using `evaluate_model` and print the accuracy.
    7. Save the model to disk using `save_model`.
    8. Load and predict a new user's class using `predict_new_user`.
    9. Print the predicted result.

**Execution Steps to Follow:**

1. All actions like build, compile, running application, running test cases will be through Command Terminal.
2. To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal
3. This editor Auto Saves the code
4. If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) 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.
5. **To launch application: python3 filename.py**
6. **To run Test cases: python3 -m unittest**

7. 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” in the right after you are done with the code.

8.