System Requirements Specification Index

For

Predict Diabetes using TensorFlow

Version 1.0

You are assigned to build a **binary classification model** that predicts whether a person is **Diabetic** or **Non-Diabetic** based on medical data. You will use the **Pima Indians Diabetes Dataset**, a popular dataset for health-related machine learning tasks.

Your tasks include:

- Data loading
- Preprocessing
- Model creation and training
- Prediction on new samples
- Reading test samples from a file

You are expected to write the code in **modular functions**, test the prediction, and print appropriate outputs.

Dataset Information: `pima-indians-diabetes.data.csv`

- Format: CSV (no header)
- Columns (in order):
    1. **Pregnancies**
    2. **Glucose**
    3. **BloodPressure**
    4. **SkinThickness**
    5. **Insulin**
    6. **BMI**
    7. **DiabetesPedigreeFunction**
    8. **Age**
    9. **Outcome** (0 = Non-Diabetic, 1 = Diabetic)

Ensure this CSV file is placed in the **same directory** as your code file.

Functional Requirements

You must implement the following functions **using the `TODO` markers** provided in the skeleton:

---

1. `load_dataset() -> pd.DataFrame`

- Load the dataset file named `pima-indians-diabetes.data.csv` using the `pandas.read_csv()` function.
- Assign the following nine column names to the DataFrame: `'Pregnancies'`, `'Glucose'`, `'BloodPressure'`, `'SkinThickness'`, `'Insulin'`, `'BMI'`, `'DiabetesPedigreeFunction'`, `'Age'`, and `'Outcome'`.

- Return the resulting DataFrame containing the loaded and labeled data.

---

2. **`preprocess_data(df) -> tuple`**

- Split the given DataFrame into two separate parts: features (`X`) and labels (`y`).
- Apply feature scaling to X using `StandardScaler`, which standardizes each feature to have approximately zero mean and unit variance (i.e., mean ≈ 0, std ≈ 1).
  Note: StandardScaler does not bound values between fixed ranges like [0, 1] or [-3, 3]; rather, it transforms features into standard normal distributions.
- Split the data into training and test sets using `train_test_split` with an 80/20 ratio.
- Return the four resulting datasets: `X_train`, `X_test`, `y_train`, and `y_test`.

---

3. **`build_model(input_dim: int) -> tf.keras.Model`**

- Construct a neural network model using `tf.keras.Sequential`.
- Add the following layers in sequence:
  - A `Dense` layer with 64 units and `'relu'` activation.
  - A second `Dense` layer with 32 units and `'relu'` activation.
  - A final `Dense` layer with 1 unit and `'sigmoid'` activation for binary classification.
- Ensure the first layer has the parameter `input_shape=(input_dim,)` to define the expected input dimensions.

---

4. **`compile_model(model: tf.keras.Model) -> None`**

- Compile the provided Keras model using the following configurations:
  - Optimizer: `'adam'`
  - Loss function: `'binary_crossentropy'`
  - Metric: `'accuracy'`
- This prepares the model for training.

---

5. **`train_model(model, X_train, y_train, X_val, y_val, epochs=20) -> History`**

- Train the compiled model using the `model.fit()` method.
- Provide the training data (`X_train`, `y_train`) and validation data (`X_val`, `y_val`).
- Set `batch_size=32`, `epochs=20`, and `verbose=0` to control training output.
- Return the training history object containing loss and accuracy metrics over epochs.

---

6. **`evaluate_model(model, X_test, y_test) -> float`**

- Evaluate the trained model using the test dataset (`X_test`, `y_test`).
- Calculate and print the model's accuracy in percentage format (e.g., `Test Accuracy: 88.45%`).
- Return the computed accuracy value as a float.

---

7. `predict_sample(model, sample) -> str`

- Reshape the input sample into the shape `(1, -1)` using `numpy.reshape`.
- Use the trained model to predict the outcome for the sample with `model.predict()`.
- Return a string indicating the prediction result:
  - Return `"Diabetic"` if the predicted output is greater than or equal to 0.5.
  - Otherwise, return `"Non-Diabetic"`.

---

8. `prepare_sample_input(raw_sample, scaler) -> np.ndarray`

- Use the same `StandardScaler` instance (from preprocessing) to transform the `raw_sample`.
- Return the scaled and transformed sample as a 1D NumPy array suitable for prediction.

---

9. `load_sample_from_file(filename="sample_input.txt") -> list or None`

- Open the file with the specified filename (default is `"sample_input.txt"`).
- Read the first line from the file and split it by commas.
- Convert the resulting string values into a list of floats.
- Return the list of float values if successful; return `None` if there is an error while reading or parsing the file.

Sample content of `sample_input.txt`:

```
5,116,74,0,0,25.6,0.201,30
```

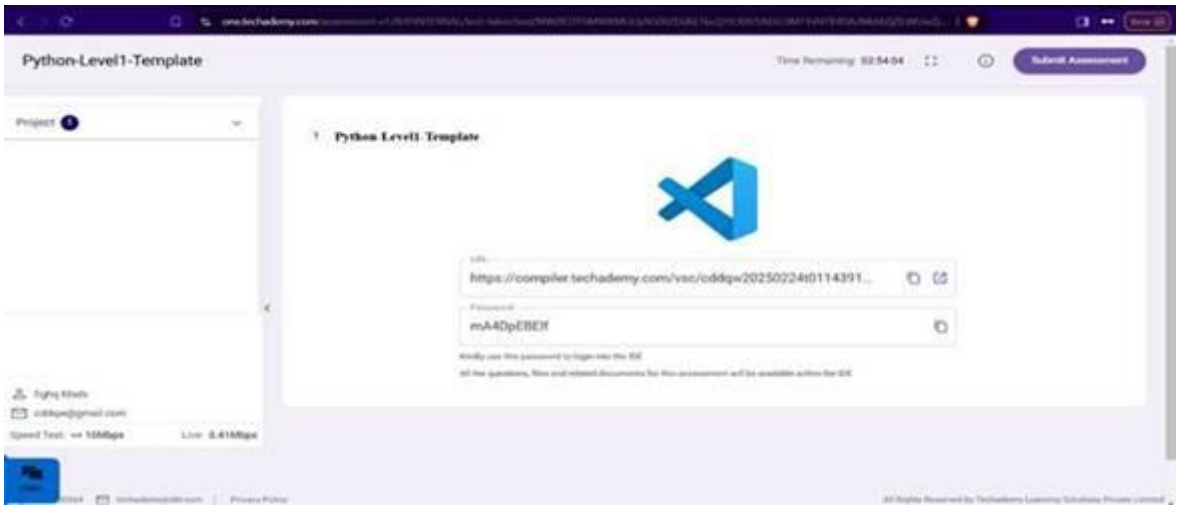
Execution Steps to Follow:

- 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)
- 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 launch application: python3 filename.py
- To run Test cases: python3 -m unittest

Screen shot to run the program

- To run the application
- python3 filename.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.