## **Practice Question 1:**

### **Task**

You are required to analyze and visualize a student performance dataset, create a predictive model using PyTorch, and manage the data using a SQLite database.

### **Subtasks**

1. **Data Processing and Visualization (12 Marks)**:
   * Load a CSV file named student\_scores.csv containing columns: StudentID, Name, MathScore, EnglishScore, and ScienceScore.
   * Visualize:
     + Average scores in each subject using a bar chart.
     + A scatter plot of MathScore vs ScienceScore.
   * Calculate and display the overall average score per student and save it as a new column OverallScore.
2. **Simple Neural Network Using PyTorch (15 Marks)**:
   * Create a fully connected neural network in PyTorch to predict the OverallScore based on MathScore, EnglishScore, and ScienceScore.
   * Train the model using a subset of the data and evaluate its accuracy.
3. **Database Management (13 Marks)**:
   * Save the processed data into an SQLite database named StudentDB.
   * Implement the following operations:
     + Display all records.
     + Retrieve and display the top 3 students based on OverallScore.
     + Clear all records from the database.

## **Solution**

### **Code**

| python  Copy code  # Import necessary libraries  import pandas as pd  import matplotlib.pyplot as plt  import sqlite3  import torch  import torch.nn as nn  import torch.optim as optim  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  # --------------------------------  # Part 1: Data Processing and Visualization  # --------------------------------  # Load data  data = pd.read\_csv('student\_scores.csv')  # Calculate Overall Score  data['OverallScore'] = data[['MathScore', 'EnglishScore', 'ScienceScore']].mean(axis=1)  # Visualization: Bar chart for average scores  subject\_means = data[['MathScore', 'EnglishScore', 'ScienceScore']].mean()  subject\_means.plot(kind='bar', title='Average Subject Scores', ylabel='Score', xlabel='Subject')  plt.show()  # Visualization: Scatter plot for Math vs Science Scores  plt.scatter(data['MathScore'], data['ScienceScore'])  plt.title('Math vs Science Scores')  plt.xlabel('Math Score')  plt.ylabel('Science Score')  plt.show()  # --------------------------------  # Part 2: Neural Network Using PyTorch  # --------------------------------  # Prepare data for the neural network  X = data[['MathScore', 'EnglishScore', 'ScienceScore']].values  y = data['OverallScore'].values  # Normalize features  scaler = StandardScaler()  X = scaler.fit\_transform(X)  # Split data  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Convert to PyTorch tensors  X\_train\_tensor = torch.tensor(X\_train, dtype=torch.float32)  y\_train\_tensor = torch.tensor(y\_train, dtype=torch.float32).unsqueeze(1)  X\_test\_tensor = torch.tensor(X\_test, dtype=torch.float32)  y\_test\_tensor = torch.tensor(y\_test, dtype=torch.float32).unsqueeze(1)  # Define the neural network  class NeuralNet(nn.Module):  def \_\_init\_\_(self):  super(NeuralNet, self).\_\_init\_\_()  self.fc1 = nn.Linear(3, 16)  self.fc2 = nn.Linear(16, 1)    def forward(self, x):  x = torch.relu(self.fc1(x))  x = self.fc2(x)  return x  model = NeuralNet()  criterion = nn.MSELoss()  optimizer = optim.Adam(model.parameters(), lr=0.01)  # Train the model  epochs = 100  for epoch in range(epochs):  model.train()  optimizer.zero\_grad()  predictions = model(X\_train\_tensor)  loss = criterion(predictions, y\_train\_tensor)  loss.backward()  optimizer.step()  if (epoch + 1) % 10 == 0:  print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')  # Evaluate the model  model.eval()  with torch.no\_grad():  test\_predictions = model(X\_test\_tensor)  test\_loss = criterion(test\_predictions, y\_test\_tensor)  print(f'Test Loss: {test\_loss.item():.4f}')  # --------------------------------  # Part 3: Database Management  # --------------------------------  # Connect to SQLite database  conn = sqlite3.connect('StudentDB.db')  cursor = conn.cursor()  # Create table  cursor.execute('''  CREATE TABLE IF NOT EXISTS Students (  StudentID INTEGER,  Name TEXT,  MathScore REAL,  EnglishScore REAL,  ScienceScore REAL,  OverallScore REAL  )  ''')  # Insert data into the database  data.to\_sql('Students', conn, if\_exists='replace', index=False)  # Display all records  print("\nAll Records:")  for row in cursor.execute('SELECT \* FROM Students'):  print(row)  # Display top 3 students based on OverallScore  print("\nTop 3 Students by OverallScore:")  for row in cursor.execute('SELECT \* FROM Students ORDER BY OverallScore DESC LIMIT 3'):  print(row)  # Clear all records  cursor.execute('DELETE FROM Students')  conn.commit()  # Close the connection  conn.close() |
| --- |

### **Explanation**

1. **Data Processing and Visualization**:
   * Loaded data from a CSV file.
   * Calculated OverallScore as the mean of the scores.
   * Visualized subject averages and a scatter plot to analyze correlations.
2. **Neural Network**:
   * A simple two-layer fully connected neural network was created using PyTorch.
   * Features (MathScore, EnglishScore, ScienceScore) were normalized.
   * The model was trained to predict the OverallScore.
3. **Database Management**:
   * Created an SQLite database and inserted processed data.
   * Implemented queries to display records and retrieve top-performing students.
   * Cleared all records as part of database management.

## **Practice Question 2:**

### **Task**

You are required to create a weather analytics system that processes and visualizes data, predicts future temperatures using a neural network, and manages data using a SQLite database.

### **Subtasks**

1. **Data Processing and Visualization (12 Marks)**:
   * Load a CSV file named weather\_data.csv with columns: City, Date, Temperature, Humidity, WindSpeed.
   * Visualize:
     + A line chart of Temperature over time for a specific city.
     + A bar chart comparing average Humidity across all cities.
   * Add a new column ComfortIndex, calculated as: ComfortIndex=Temperature−0.55×(1−Humidity/100)×(Temperature−14.5)\text{ComfortIndex} = \text{Temperature} - 0.55 \times (1 - \text{Humidity}/100) \times (\text{Temperature} - 14.5)ComfortIndex=Temperature−0.55×(1−Humidity/100)×(Temperature−14.5)
2. **Neural Network Using PyTorch (15 Marks)**:
   * Create a neural network to predict Temperature based on Humidity and WindSpeed.
   * Train the model using 80% of the data and test it on the remaining 20%.
   * Evaluate the model's accuracy using Mean Squared Error (MSE).
3. **Database Management (13 Marks)**:
   * Save the processed data into an SQLite database named WeatherDB.
   * Implement the following operations:
     + Retrieve temperature trends for a specific city over a given time range.
     + Retrieve cities with average Temperature above a user-specified threshold.
     + Clear all records from the database.

## **Solution**

### **Code**

| python  # Import necessary libraries  import pandas as pd  import matplotlib.pyplot as plt  import sqlite3  import torch  import torch.nn as nn  import torch.optim as optim  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  # --------------------------------  # Part 1: Data Processing and Visualization  # --------------------------------  # Load the data  data = pd.read\_csv('weather\_data.csv', parse\_dates=['Date'])  # Calculate ComfortIndex  data['ComfortIndex'] = data['Temperature'] - 0.55 \* (1 - data['Humidity'] / 100) \* (data['Temperature'] - 14.5)  # Visualization: Temperature over time for a specific city  city = 'New York' # Example city  city\_data = data[data['City'] == city]  plt.plot(city\_data['Date'], city\_data['Temperature'], label='Temperature')  plt.title(f'Temperature Over Time in {city}')  plt.xlabel('Date')  plt.ylabel('Temperature')  plt.legend()  plt.show()  # Visualization: Average humidity across all cities  avg\_humidity = data.groupby('City')['Humidity'].mean()  avg\_humidity.plot(kind='bar', title='Average Humidity by City', ylabel='Humidity', xlabel='City')  plt.show()  # --------------------------------  # Part 2: Neural Network Using PyTorch  # --------------------------------  # Prepare data for the neural network  X = data[['Humidity', 'WindSpeed']].values  y = data['Temperature'].values  # Normalize features  scaler = StandardScaler()  X = scaler.fit\_transform(X)  # Split data  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Convert to PyTorch tensors  X\_train\_tensor = torch.tensor(X\_train, dtype=torch.float32)  y\_train\_tensor = torch.tensor(y\_train, dtype=torch.float32).unsqueeze(1)  X\_test\_tensor = torch.tensor(X\_test, dtype=torch.float32)  y\_test\_tensor = torch.tensor(y\_test, dtype=torch.float32).unsqueeze(1)  # Define the neural network  class NeuralNet(nn.Module):  def \_\_init\_\_(self):  super(NeuralNet, self).\_\_init\_\_()  self.fc1 = nn.Linear(2, 16)  self.fc2 = nn.Linear(16, 1)    def forward(self, x):  x = torch.relu(self.fc1(x))  x = self.fc2(x)  return x  model = NeuralNet()  criterion = nn.MSELoss()  optimizer = optim.Adam(model.parameters(), lr=0.01)  # Train the model  epochs = 100  for epoch in range(epochs):  model.train()  optimizer.zero\_grad()  predictions = model(X\_train\_tensor)  loss = criterion(predictions, y\_train\_tensor)  loss.backward()  optimizer.step()  if (epoch + 1) % 10 == 0:  print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')  # Evaluate the model  model.eval()  with torch.no\_grad():  test\_predictions = model(X\_test\_tensor)  test\_loss = criterion(test\_predictions, y\_test\_tensor)  print(f'Test Loss (MSE): {test\_loss.item():.4f}')  # --------------------------------  # Part 3: Database Management  # --------------------------------  # Connect to SQLite database  conn = sqlite3.connect('WeatherDB.db')  cursor = conn.cursor()  # Create table  cursor.execute('''  CREATE TABLE IF NOT EXISTS WeatherData (  City TEXT,  Date TEXT,  Temperature REAL,  Humidity REAL,  WindSpeed REAL,  ComfortIndex REAL  )  ''')  # Insert data into the database  data.to\_sql('WeatherData', conn, if\_exists='replace', index=False)  # Query 1: Retrieve temperature trends for a specific city over a given date range  city = 'New York'  start\_date = '2023-01-01'  end\_date = '2023-12-31'  print(f"\nTemperature Trends in {city} from {start\_date} to {end\_date}:")  query = f'''  SELECT Date, Temperature FROM WeatherData  WHERE City = ? AND Date BETWEEN ? AND ?  ORDER BY Date  '''  for row in cursor.execute(query, (city, start\_date, end\_date)):  print(row)  # Query 2: Retrieve cities with average Temperature above a threshold  threshold = 25  print(f"\nCities with Average Temperature Above {threshold}:")  query = f'''  SELECT City, AVG(Temperature) as AvgTemp FROM WeatherData  GROUP BY City  HAVING AvgTemp > ?  '''  for row in cursor.execute(query, (threshold,)):  print(row)  # Clear all records  cursor.execute('DELETE FROM WeatherData')  conn.commit()  # Close the connection  conn.close() |
| --- |

### **Explanation**

1. **Data Processing and Visualization**:
   * The dataset is loaded and processed by adding a ComfortIndex column based on a formula that combines temperature and humidity.
   * Visualizations include a line chart (temperature over time for a city) and a bar chart (average humidity across cities).
2. **Neural Network**:
   * A fully connected neural network is created in PyTorch with two input features (Humidity, WindSpeed) and one output (Temperature).
   * The model is trained and evaluated using Mean Squared Error (MSE) to measure prediction accuracy.
3. **Database Management**:
   * The processed data is stored in a SQLite database.
   * Complex queries retrieve data based on conditions like temperature trends over time and cities exceeding a temperature threshold.