Task-1:

Step 1: Importing Libraries

Capabilities:

- Imports the necessary libraries for building and training a neural network model for MNIST digit classification.

Installation:

- Ensure you have TensorFlow installed on your system.

Usage:

- The `import` statements bring the required modules and functions into the current namespace.

Important Details:

- TensorFlow is a popular open-source machine learning library widely used for building and deploying neural networks.
- Keras is a high-level API for TensorFlow that simplifies the process of creating and training neural networks.
- The `mnist` dataset provides a collection of handwritten digits for training and evaluating classification models.
- The `to_categorical` function converts class labels into a one-hot encoded format suitable for neural network training.

Step 2: Loading and Preprocessing the MNIST Dataset

Capabilities:

- Loads the MNIST dataset, consisting of handwritten digits, and splits it into training and testing sets.
- Preprocesses the training and testing images by reshaping them into a 4D tensor format and normalizing their values between 0 and 1.

- Converts class labels into a one-hot encoded format suitable for neural network training.

Installation:

- Ensure you have TensorFlow installed on your system.

Usage:

1. The `mnist.load_data()` function loads the MNIST dataset into two pairs: training data (training images and training labels) and testing data (testing images and testing labels).
2. The `reshape` function converts the training and testing images into a 4D tensor format, where the first dimension represents the number of samples, the second and third dimensions represent the height and width of the images, and the fourth dimension represents the number of channels (1 for grayscale images).
3. The `astype` function converts the training and testing images from unsigned 8-bit integers (0 to 255) to floating-point values between 0 and 1. This normalization step helps improve the performance of the neural network model.
4. The `to_categorical` function converts the class labels from integers (0 to 9) into a one-hot encoded format. For example, the label 5 would be represented as a vector of zeros with a 1 in the fifth position. This format is more suitable for neural network training.

Important Details:

- The MNIST dataset is a standard benchmark dataset for evaluating handwritten digit classification models.
- Normalizing image values helps ensure that the model learns from features rather than absolute pixel intensities.
- One-hot encoding is a common way to represent categorical data in neural network training.

Step 3: Building the Neural Network Model

Capabilities:

- Creates a sequential neural network model with a specific architecture for MNIST digit classification.

- Defines the network layers, including convolutional layers for feature extraction, pooling layers for reducing dimensionality, and dense layers for classification.

Model Architecture:

Input Layer (Conv2D):
Number of Filters (Kernels): 32
Kernel Size: (3, 3)
Activation Function: ReLU (Rectified Linear Unit)
Input Shape: (28, 28, 1) - This is the shape of a single MNIST image. The "1" represents a single channel because MNIST images are grayscale.
The first Conv2D layer acts as a feature extractor. It uses 32 filters to convolve over the input image, detecting low-level features like edges and simple patterns.

MaxPooling Layer:
Pool Size: (2, 2)
Max pooling is applied after each convolutional layer to reduce the spatial dimensions of the representation and decrease the computational complexity. It retains the most important information from the previous layer.

Second Convolutional Layer (Conv2D):
Number of Filters (Kernels): 64
Kernel Size: (3, 3)
Activation Function: ReLU
The second Conv2D layer further extracts higher-level features using 64 filters. The network learns to recognize more complex patterns.

MaxPooling Layer:
Pool Size: (2, 2)
Another max pooling layer follows the second convolutional layer to reduce spatial dimensions.

Third Convolutional Layer (Conv2D):
Number of Filters (Kernels): 64
Kernel Size: (3, 3)
Activation Function: ReLU
The third Conv2D layer continues to extract even more complex features.

Flatten Layer:
This layer is used to flatten the output from the previous layer into a 1D array. It prepares the data for the fully connected (dense) layers.

Dense (Fully Connected) Layer:
Number of Neurons: 64
Activation Function: ReLU
The first dense layer acts as a classifier, making decisions based on the extracted features. It introduces non-linearity to the model.

Output Layer (Dense):
Number of Neurons: 10 (because there are 10 classes in the MNIST dataset)
Activation Function: Softmax
The output layer produces a probability distribution over the 10 classes using the softmax activation function. Each neuron in the output layer represents the probability of the corresponding digit (0-9).

Task-2:

Step 1: Creating the Database

Capabilities:

- This Step creates a SQLite database named `telecom_customers.db` to store telecom customer data.
- It defines a table named `telecom_customers` with seven columns: `customer_id`, `first_name`, `last_name`, `phone_number`, `email_address`, `plan_type`, and `activation_date`.
- The `customer_id` column serves as the primary key, uniquely identifying each customer record.
- Data types are appropriately chosen for each column to ensure data integrity and consistency.

Installation:

- Ensure Python 3.x is installed on your system.
- Install the SQLite3 module using the command `pip install sqlite3`.

Usage:

- The provided code snippet directly creates the database and table structure.
- No further installation or configuration is required.

Other Important Details:

- The database is designed to efficiently store and manage telecom customer data.
- It facilitates easy retrieval, addition, update, and deletion of customer information.
- The database structure adheres to the principles of data normalization and entity-relationship modeling.

Step 2: Adding Customers

Capabilities:

- This Step defines a function `add_customer()` that inserts new customer records into the `telecom_customers` table.
- It takes six arguments as input: `first_name`, `last_name`, `phone_number`, `email_address`, `plan_type`, and `activation_date`.
- It stores the provided customer information into the corresponding columns of the database table.

Installation:

- Follow the installation instructions for Step 1.

Usage:

- Import the `sqlite3` module at the beginning of your Python script.
- Call the `add_customer()` function with the desired customer information.
- The function will insert the provided data into the database.

Other Important Details:

- The function ensures data integrity by using proper data types and atomic operations.
- It adds multiple customer records in a single operation, improving efficiency.

Step 3: Retrieving Customers

Capabilities:

- This Step defines a function `get_all_customers()` that retrieves all customer records from the `telecom_customers` table.
- It executes a SQL query to fetch all customer data.
- It returns a list of tuples, where each tuple represents a customer record.

Installation:

- Follow the installation instructions for Step 1.

Usage:

- Import the `sqlite3` module at the beginning of your Python script.
- Call the `get_all_customers()` function to retrieve all customer data.
- The returned list can be processed or displayed as needed.

Other Important Details:

- The function efficiently fetches all customer records from the database.
- It returns the data in a structured format, allowing for easy processing and analysis.

Step 4: Updating Customer Plan Type

Capabilities:

- This Step defines a function `update_plan_type()` that updates the `plan_type` column for a specific customer in the `telecom_customers` table.
- It takes two arguments as input: `customer_id` and `new_plan_type`.
- It identifies the customer record using the provided `customer_id`.
- It updates the `plan_type` column with the specified `new_plan_type` value.

Installation:

- Follow the installation instructions for Step 1.

Usage:

- Import the `sqlite3` module at the beginning of your Python script.
- Call the `update_plan_type()` function with the customer ID and the new plan type.
- The function will update the plan type for the specified customer in the database.

Other Important Details:

- The function enables modification of customer plan types based on their unique IDs.
- It ensures data integrity by using atomic operations and proper data types.

Step 5: Deleting Customers

Capabilities:

- This Step defines a function `delete_customer()` that removes a customer record from the `telecom_customers` table.
- It takes a single argument as input: `customer_id`.
- It identifies the customer record using the provided `customer_id`.
- It deletes the identified customer record from the database.

Installation:

- Follow the installation instructions for Step 1.

Usage:

- Import the `sqlite3` module at the beginning of your Python script.
- Call the `delete_customer()` function with the customer ID of the record to be removed.
- The function will delete the specified customer record from the database.

Other Important Details:

- The function enables selective deletion of customer records based

Task-3:

This Python script demonstrates how to use the Google Maps Geocoding API to retrieve the geographic coordinates (latitude and longitude) of a location based on its address. The script is organized into a modular structure, making it easy to understand and use.

Requirements
Python 3.x
requests library (install using pip install requests)
Google Maps API Key
Installation
Install Python:

Ensure that you have Python 3.x installed on your system. If not, you can download it from python.org.
Install Requests Library:

Open a terminal or command prompt and run the following command to install the required requests library.

Google Maps API Key:

Obtain a Google Maps API Key from the Google Cloud Console.
Enable the "Geocoding API" for your project.
Replace the placeholder '#bdhskyldjjw12ndwjlqlkjhu' in the script with your actual API key.
Usage
Run the Script:

Save the provided script to a file, for example, geocoding_script.py.
Open a terminal or command prompt and navigate to the directory where the script is saved.

Script Structure
1. Import Statements:

The script starts by importing the requests library, which is used for making HTTP requests.
2. Global Constants:
The API key and the Google Maps Geocoding API endpoint are defined as global constants.
3. Function Definition:
The geocode_address function is defined to encapsulate the process of making a request to the Google Maps Geocoding API and processing the response.
It takes an address as input and returns a tuple of latitude and longitude if successful, or None otherwise.
4. API Request:
Inside the geocode_address function, the parameters for the API request are prepared, including the address and the API key.
The script then makes a GET request to the Google Maps Geocoding API using the requests.get method.
5. Data Processing:
The script checks if the API request was successful by examining the 'status' field in the JSON response.
If the response indicates success, the latitude and longitude are extracted from the JSON data.
If there is an error, it prints an error message along with the status and, if available, an error message from the API.
6. Main Block:
The main block of the script demonstrates how to use the geocode_address function by providing an example address and printing the result.

Submitter By:
TAREQ BIN AHAMMED
Applicant for Data Scientist