4.1

```
# 4.1 Nguyen Viet QuangS
import os
import json
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import accuracy_score, mean_absolute_error, mean_squared_error
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, LSTM, Conv1D
import tensorflow as tf

# Suppress TensorFlow warnings
tf.get_logger().setLevel('ERROR')
```

```
# Helper function to ensure directory exists
def create_directory(path):
    if not os.path.exists(path):
        os.makedirs(path)
        print(f"Created directory: {path}")
    else:
        print(f"Directory already exists: {path}")
```

a. Build a dataset with 1000 persons and 4 features (jobs, age, height, weight) and store in C:\DATA\data_4.1.csv

```python
# --- (a) Dataset with 1000 persons and 4 features ---
def generate_dataset():
    """Generates a dataset of 1000 persons and saves it to a CSV file."""
    print("--- (a) Generating dataset ---")
    jobs = ['Engineer', 'Doctor', 'Teacher', 'Artist', 'Programmer', 'Lawyer', 'Accountant', 'Student', 'Scientist']
    num_persons = 1000

    data = {
        'age': np.random.randint(20, 60, num_persons),
        'height': np.random.normal(loc=170, scale=10, size=num_persons).round(2),
        'weight': np.random.normal(loc=70, scale=15, size=num_persons).round(2),
        'jobs': np.random.choice(jobs, num_persons)
    }

    df = pd.DataFrame(data)

    # Ensure height and weight are non-negative
    df['height'] = df['height'].apply(lambda x: max(150, x))
    df['weight'] = df['weight'].apply(lambda x: max(40, x))

    # Define the file path and ensure the directory exists
    file_path = r'C:\DATA\data_4.1.csv'
    create_directory(os.path.dirname(file_path))

    df.to_csv(file_path, index=False)
    print(f"Dataset generated and saved to {file_path}")
    return df
```

b. Show the distribution of the dataset

```python
# --- (b) Show the distribution of the dataset ---
def show_distribution(df):
    """Shows the basic distribution of the dataset."""
    print("\n--- (b) Dataset Distribution ---")
    print("DataFrame Info:")
    df.info()
    print("\nDescriptive Statistics:")
    print(df.describe().round(2))
```

c. Using 5 basic ML models and using BMI to classifying and predicting overweight, underweight, normal with respect to age and job

```python
# --- (c) Basic ML models for classification ---
def train_basic_ml_models(df):
    """Trains 5 basic ML models to classify BMI categories."""
    print("\n--- (c) Training 5 Basic ML Models ---")

    # Calculate BMI and create the target variable
    df['bmi'] = df['weight'] / (df['height'] / 100)**2
    def classify_bmi(bmi):
        if bmi < 18.5:
            return 'Underweight'
        elif bmi >= 25.0:
            return 'Overweight'
        else:
            return 'Normal'
    df['bmi_category'] = df['bmi'].apply(classify_bmi)

    # Define features and target
    X = df[['age', 'jobs']]
    y = df['bmi_category']

    # Preprocessing: One-hot encode 'jobs' and scale 'age'
    preprocessor = ColumnTransformer(
        transformers=[
            ('cat', OneHotEncoder(handle_unknown='ignore'), ['jobs']),
            ('num', StandardScaler(), ['age'])
        ],
        remainder='passthrough'
    )
```

```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit preprocessor on training data
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)

models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42),
    'Support Vector Machine (SVC)': SVC(random_state=42)
}

results = {}
for name, model in models.items():
    print(f"Training {name}...")
    model.fit(X_train_processed, y_train)
    y_pred = model.predict(X_test_processed)
    results[name] = {
        'model': model,
        'y_test': y_test,
        'y_pred': y_pred
    }

return results, preprocessor, df
```

d. Build models CNN, RNN, LSTM with 5 layers for classification and prediction problem in c

```
# --- (d) Build CNN, RNN, LSTM models ---
def build_deep_learning_models(X_train_processed, y_train, preprocessor):
    """
    Builds and trains CNN, RNN, and LSTM models for the classification task.

    Note: These models are generally designed for sequential or spatial data.
    For this tabular data, we reshape the input to be a 1D sequence for each sample.
    """
    print("\n--- (d) Building and Training Deep Learning Models ---")

    # Encode target labels
    label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(y_train)
    num_classes = len(label_encoder.classes_)

    # Reshape data for deep learning models
    # The shape should be (samples, features, 1) for Conv1D, LSTM, RNN
    X_train_reshaped = X_train_processed.toarray() if hasattr(X_train_processed, 'toarray') else X_train_processed
    X_train_reshaped = X_train_reshaped.reshape(X_train_reshaped.shape[0], X_train_reshaped.shape[1], 1)

    # Define the input shape
    input_shape = (X_train_reshaped.shape[1], 1)
```

```python
# --- CNN Model ---
def build_cnn():
    model = Sequential([
        tf.keras.Input(shape=input_shape),
        Conv1D(filters=32, kernel_size=3, activation='relu'),
        Dropout(0.2),
        Conv1D(filters=64, kernel_size=3, activation='relu'),
        Dropout(0.2),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

```python
# --- LSTM Model ---
def build_lstm():
    model = Sequential([
        tf.keras.Input(shape=input_shape),
        LSTM(64, return_sequences=True),
        Dropout(0.2),
        LSTM(64),
        Dropout(0.2),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

```python
# --- RNN Model (using SimpleRNN) ---
def build_rnn():
    model = Sequential([
        tf.keras.Input(shape=input_shape),
        tf.keras.layers.SimpleRNN(64, return_sequences=True),
        Dropout(0.2),
        tf.keras.layers.SimpleRNN(64),
        Dropout(0.2),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

```
deep_models = {
    'CNN': build_cnn(),
    'LSTM': build_lstm(),
    'RNN': build_rnn()
}

dl_results = {}
for name, model in deep_models.items():
    print(f"\nTraining {name}...")
    model.fit(X_train_reshaped, y_encoded, epochs=10, batch_size=32, verbose=0)
    dl_results[name] = {'model': model, 'label_encoder': label_encoder}

# Create the health guide JSON file
create_health_guide_kb()

return dl_results, preprocessor
```

Function to create the health guide knowledge base

```
# Function to create the health guide knowledge base
def create_health_guide_kb():
    """Builds a simple JSON knowledge base for health advice."""
    health_guide = {
        "Underweight": {
            "title": "Health Guide: Underweight",
            "advice": [
                "Increase your calorie intake with nutrient-dense foods like whole grains, healthy fats, and lean proteins.",
                "Eat more frequent meals throughout the day.",
                "Incorporate strength training exercises to build muscle mass.",
                "Consult a nutritionist or doctor for a personalized plan."
            ]
        },
        "Normal": {
            "title": "Health Guide: Normal Weight",
            "advice": [
                "Maintain a balanced diet rich in fruits, vegetables, and lean proteins.",
                "Engage in regular physical activity, including both cardio and strength training.",
                "Stay hydrated and get sufficient sleep.",
                "Continue to monitor your health and well-being."
            ]
        },
        "Overweight": {
            "title": "Health Guide: Overweight",
            "advice": [
                "Focus on a diet with a calorie deficit by reducing intake of processed foods and sugary drinks.",
                "Increase daily physical activity, such as walking, jogging, or cycling.",
                "Choose smaller portion sizes and eat mindfully.",
                "Seek advice from a healthcare professional to set realistic weight loss goals."
            ]
        }
    }
    file_path = r'C:\DATA\kb_healthGuide.json'
    create_directory(os.path.dirname(file_path))
    with open(file_path, 'w') as f:
        json.dump(health_guide, f, indent=4)
    print(f"\nHealth guide knowledge base created and saved to {file_path}")
```

e. Compare and evaluate models given in c. and d. with metrics accuracy, MAE, MSE, RMSE

```python
# --- (e) Compare and evaluate models ---
def evaluate_models(ml_results, dl_results, X_test, y_test, preprocessor):
    """Compares and evaluates all models with various metrics."""
    print("\n--- (e) Model Comparison and Evaluation ---")

    # Prepare data for deep learning models
    label_encoder = dl_results['CNN']['label_encoder']
    y_test_encoded = label_encoder.transform(y_test)
    X_test_processed = preprocessor.transform(X_test)
    X_test_reshaped = X_test_processed.toarray() if hasattr(X_test_processed, 'toarray') else X_test_processed
    X_test_reshaped = X_test_reshaped.reshape(X_test_reshaped.shape[0], X_test_reshaped.shape[1], 1)

    all_results = {}

    # Evaluate ML models
    for name, res in ml_results.items():
        y_test_num = label_encoder.transform(res['y_test'])
        y_pred_num = label_encoder.transform(res['y_pred'])
        all_results[name] = {
            'Accuracy': accuracy_score(res['y_test'], res['y_pred']),
            'MAE': mean_absolute_error(y_test_num, y_pred_num),
            'MSE': mean_squared_error(y_test_num, y_pred_num),
            'RMSE': np.sqrt(mean_squared_error(y_test_num, y_pred_num))
        }
```

```python
    # Evaluate Deep Learning models
    for name, res in dl_results.items():
        y_pred_probs = res['model'].predict(X_test_reshaped, verbose=0)
        y_pred_encoded = np.argmax(y_pred_probs, axis=1)
        y_pred = label_encoder.inverse_transform(y_pred_encoded)

        y_pred_num = y_pred_encoded
        y_test_num = y_test_encoded

        all_results[name] = {
            'Accuracy': accuracy_score(y_test, y_pred),
            'MAE': mean_absolute_error(y_test_num, y_pred_num),
            'MSE': mean_squared_error(y_test_num, y_pred_num),
            'RMSE': np.sqrt(mean_squared_error(y_test_num, y_pred_num))
        }


    # Create and display evaluation table
    evaluation_df = pd.DataFrame(all_results).T.round(3)
    print(evaluation_df)

    # Find the best model based on accuracy
    best_model_name = evaluation_df['Accuracy'].idxmax()
    print(f"\nBest model based on Accuracy: {best_model_name}")

    if best_model_name in ml_results:
        best_model = ml_results[best_model_name]['model']
    else:
        best_model = dl_results[best_model_name]['model']

    return best_model, best_model_name, preprocessor, label_encoder
```

f. Visualize with >=5 various types

```python
# --- (f) Visualize with >=5 various types ---
def visualize_data(df, ml_results):
    """Generates various plots to visualize the data and model results."""
    print("\n--- (f) Data Visualization ---")

    plt.style.use('seaborn-v0_8-whitegrid')
    fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(15, 18))
    fig.suptitle('Dataset and Model Evaluation Visualizations', fontsize=20, y=1.02)

    # Plot 1: Age Distribution
    sns.histplot(df['age'], bins=15, kde=True, ax=axes[0, 0], color='skyblue')
    axes[0, 0].set_title('Age Distribution')
    axes[0, 0].set_xlabel('Age (years)')
    axes[0, 0].set_ylabel('Count')

    # Plot 2: BMI Category Distribution
    sns.countplot(x='bmi_category', data=df, ax=axes[0, 1], palette='viridis', hue='bmi_category', legend=False)
    axes[0, 1].set_title('BMI Category Distribution')
    axes[0, 1].set_xlabel('BMI Category')
    axes[0, 1].set_ylabel('Count')

    # Plot 3: Scatter plot of Age vs. BMI
    sns.scatterplot(x='age', y='bmi', hue='bmi_category', data=df, ax=axes[1, 0], palette='coolwarm')
    axes[1, 0].set_title('BMI vs. Age')
    axes[1, 0].set_xlabel('Age (years)')
    axes[1, 0].set_ylabel('BMI')
```

```python
# Plot 4: Job Distribution
sns.countplot(y='jobs', data=df, ax=axes[1, 1], palette='plasma', order=df['jobs'].value_counts().index, hue='jobs', legend=False)
axes[1, 1].set_title('Job Distribution')
axes[1, 1].set_xlabel('Count')
axes[1, 1].set_ylabel('Job')

# Plot 5: Height vs. Weight Scatter Plot with BMI Categories
sns.scatterplot(x='height', y='weight', hue='bmi_category', data=df, ax=axes[2, 0], palette='magma')
axes[2, 0].set_title('Height vs. Weight')
axes[2, 0].set_xlabel('Height (cm)')
axes[2, 0].set_ylabel('Weight (kg)')

# Plot 6: Confusion Matrix for a sample model (e.g., Random Forest)
best_ml_model_name = max(ml_results, key=lambda k: accuracy_score(ml_results[k]['y_test'], ml_results[k]['y_pred']))
y_pred = ml_results[best_ml_model_name]['y_pred']
y_test = ml_results[best_ml_model_name]['y_test']

confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
sns.heatmap(confusion_matrix, annot=True, fmt='d', cmap='Blues', ax=axes[2, 1])
axes[2, 1].set_title(f'Confusion Matrix for {best_ml_model_name}')

plt.tight_layout(pad=3.0)
plt.show()
```

g. Deploy the best model so that user enters: job, age, height, weight and output is underweight, overweight or normal and give a guide for health.

```python
# --- (g) Deploy the best model ---
def deploy_model(best_model, best_model_name, preprocessor, label_encoder):
    """
    Deploys the best model, takes user input, makes a prediction,
    and provides a health guide.
    """
    print(f"\n" + "="*50)
    print(f"--- (g) Deploying the Best Model: {best_model_name} ---")
    print("="*50)

    # Load the health guide knowledge base
    try:
        with open(r'C:\DATA\kb_healthGuide.json', 'r') as f:
            health_guide_kb = json.load(f)
    except FileNotFoundError:
        print("Health guide file not found. Please run the full script to generate it.")
        return
```

```python
def get_user_input():
    """Prompts the user for input with validation."""
    print("\nPlease enter your details to get a health analysis.")

    while True:
        try:
            age = int(input("Enter your age (20-60 years): "))
            if not (20 <= age <= 60):
                print("Age must be between 20 and 60. Please try again.")
                continue
            break
        except ValueError:
            print("Invalid input. Please enter a number for age.")

    jobs = ['Engineer', 'Doctor', 'Teacher', 'Artist', 'Programmer', 'Lawyer', 'Accountant', 'Student', 'Scientist']
    print(f"Available jobs: {', '.join(jobs)}")
    while True:
        job = input("Enter your job title: ").strip()
        if job not in jobs:
            print("Invalid job title. Please choose from the list.")
            continue
        break

    while True:
        try:
            height = float(input("Enter your height in cm: "))
            if height <= 0:
                print("Height must be a positive number. Please try again.")
                continue
            break
        except ValueError:
            print("Invalid input. Please enter a number for height.")

    while True:
        try:
            weight = float(input("Enter your weight in kg: "))
            if weight <= 0:
                print("Weight must be a positive number. Please try again.")
                continue
            break
        except ValueError:
            print("Invalid input. Please enter a number for weight.")

    return job, age, height, weight
```

```python
def predict_and_advise(job, age, height, weight):
    """
    Takes user input, makes a BMI prediction, and provides health advice.
    """
    # Calculate BMI
    bmi = weight / (height / 100)**2

    # Preprocess user input
    user_data = pd.DataFrame([[job, age]], columns=['jobs', 'age'])
    user_data_processed = preprocessor.transform(user_data)

    # Make prediction
    if best_model_name in ['CNN', 'RNN', 'LSTM']:
        # Convert to dense array and then reshape for deep learning models
        user_data_processed = user_data_processed.toarray()
        user_data_processed = user_data_processed.reshape(1, user_data_processed.shape[1], 1)
        prediction_probs = best_model.predict(user_data_processed, verbose=0)
        prediction_encoded = np.argmax(prediction_probs, axis=1)
        predicted_category = label_encoder.inverse_transform(prediction_encoded)[0]
    else:
        predicted_category = best_model.predict(user_data_processed)[0]
```

```python
# Get the actual BMI category
def classify_bmi(bmi):
    if bmi < 18.5:
        return 'Underweight'
    elif bmi >= 25.0:
        return 'Overweight'
    else:
        return 'Normal'
actual_category = classify_bmi(bmi)

print("\n" + "-"*30)
print("--- Your Health Analysis ---")
print("-"*30)
print(f"Based on your inputs:")
print(f"  Age: {age} years")
print(f"  Job: {job}")
print(f"  Height: {height} cm")
print(f"  Weight: {weight} kg")
print("\n" + "-"*30)
print(f"Your calculated BMI is: {bmi:.2f}")
print(f"This corresponds to the category: {actual_category}")
print("-" * 30)
```

```
    # Display the predicted category and health guide
    print(f"The model's prediction is: {predicted_category}")

    advice = health_guide_kb.get(predicted_category, {"title": "No Advice Found", "advice": []})
    print(f"\n" + "-"*30)
    print(f"{advice['title']}:")
    print("-" * 30)
    for item in advice['advice']:
        print(f"• {item}")
    print("\n" + "="*50)

# Loop for user input
while True:
    job_input, age_input, height_input, weight_input = get_user_input()
    predict_and_advise(job=job_input, age=age_input, height=height_input, weight=weight_input)

    another = input("Would you like to analyze another person? (yes/no): ").lower()
    if another != 'yes':
        break
```

Main script execution block

```
            break
# Main script execution block
if __name__ == "__main__":
    # a. Generate dataset
    df = generate_dataset()

    # b. Show dataset distribution
    show_distribution(df)

    # c. Train basic ML models
    ml_results, preprocessor, df_with_bmi = train_basic_ml_models(df)

    # d. Build and train deep learning models & create health guide KB
    X_train_processed = preprocessor.fit_transform(df_with_bmi[['age', 'jobs']])
    y_train = df_with_bmi['bmi_category']
    dl_results, _ = build_deep_learning_models(X_train_processed, y_train, preprocessor)

    # e. Compare and evaluate all models
    X_test = df_with_bmi[['age', 'jobs']].loc[ml_results['Random Forest']['y_test'].index]
    y_test = ml_results['Random Forest']['y_test']
    best_model, best_model_name, preprocessor, label_encoder = evaluate_models(
        ml_results, dl_results, X_test, y_test, preprocessor
    )

    # f. Visualize
    visualize_data(df_with_bmi, ml_results)

    # g. Deploy the best model
    deploy_model(best_model, best_model_name, preprocessor, label_encoder)
```

Output

```
--- (a) Generating dataset ---
Directory already exists: C:\DATA
Dataset generated and saved to C:\DATA\data_4.1.csv

--- (b) Dataset Distribution ---
DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   age     1000 non-null   int32
 1   height  1000 non-null   float64
 2   weight  1000 non-null   float64
 3   jobs    1000 non-null   object
dtypes: float64(2), int32(1), object(1)
memory usage: 27.5+ KB
```

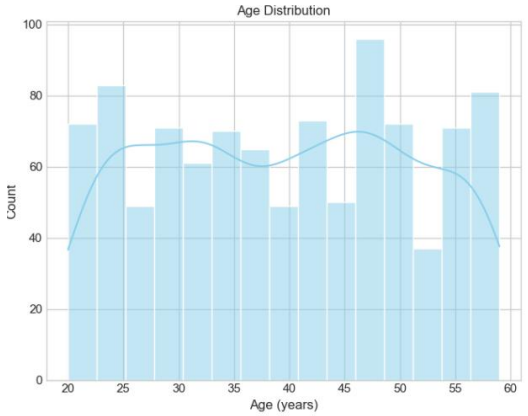
```
Descriptive Statistics:
              age   height   weight
count  1000.00  1000.00  1000.00
mean     39.43   170.39    70.68
std      11.55    10.11    14.76
min      20.00   150.00    40.00
25%      29.75   162.89    59.66
50%      40.00   170.18    70.50
75%      49.00   177.37    80.84
max      59.00   200.71   115.62


--- (c) Training 5 Basic ML Models ---
Training Logistic Regression...
Training Decision Tree...
Training Random Forest...
Training Gradient Boosting...
Training Support Vector Machine (SVC)...
```

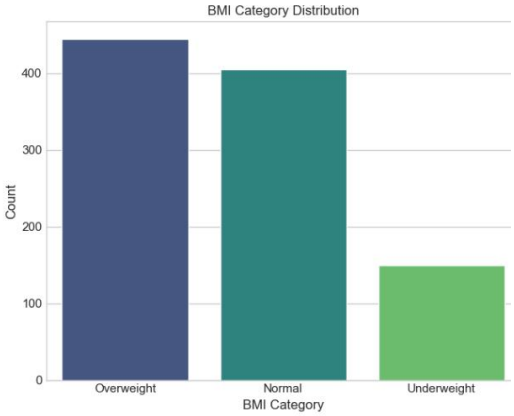
```
--- (d) Building and Training Deep Learning Models ---

Training CNN...

Training LSTM...

Training RNN...
Directory already exists: C:\DATA

Health guide knowledge base created and saved to C:\DATA\kb_healthGuide.json
```

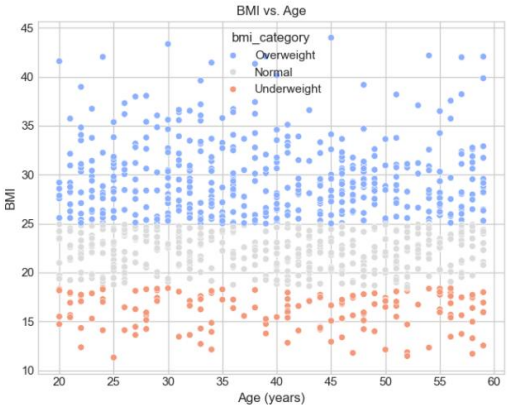
```
--- (e) Model Comparison and Evaluation ---
                              Accuracy    MAE    MSE   RMSE
Logistic Regression              0.465  0.575  0.655  0.809
Decision Tree                    0.460  0.610  0.750  0.866
Random Forest                    0.440  0.660  0.860  0.927
Gradient Boosting                0.430  0.635  0.765  0.875
Support Vector Machine (SVC)     0.435  0.600  0.670  0.819
CNN                              0.455  0.560  0.590  0.768
LSTM                             0.430  0.570  0.570  0.755
RNN                              0.440  0.650  0.830  0.911

Best model based on Accuracy: Logistic Regression
```

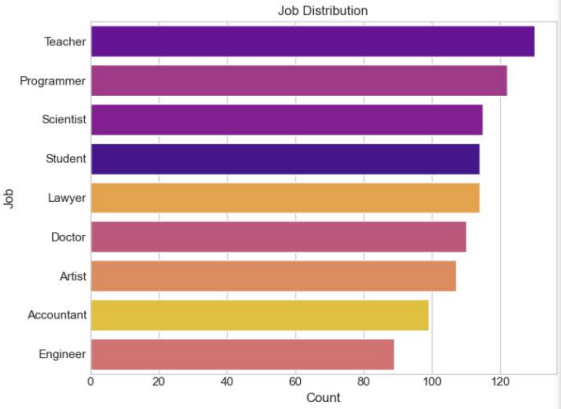
```
--- (f) Data Visualization ---
```

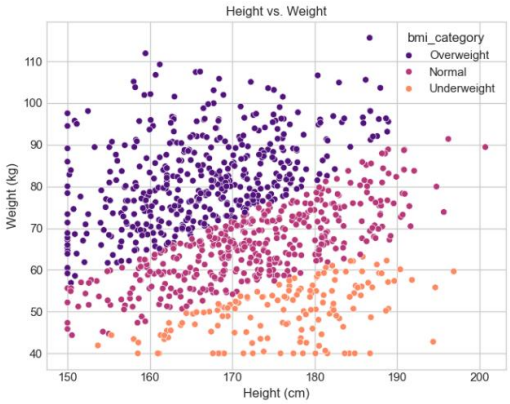
Dataset and Model Evaluation Visualizations

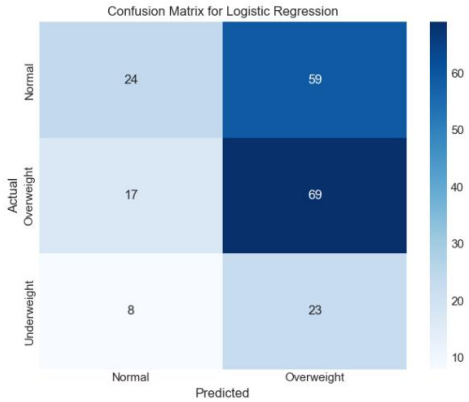













```
==================================================
--- (g) Deploying the Best Model: Logistic Regression ---
==================================================

Please enter your details to get a health analysis.
Enter your age (20-60 years):  30
Available jobs: Engineer, Doctor, Teacher, Artist, Programmer, Lawyer, Accountant, Student, Scientist
Enter your job title:  Doctor
Enter your height in cm:  160
Enter your weight in kg:  55

------------------------------
--- Your Health Analysis ---
------------------------------
Based on your inputs:
  Age: 30 years
  Job: Doctor
  Height: 160.0 cm
  Weight: 55.0 kg

------------------------------
Your calculated BMI is: 21.48
This corresponds to the category: Normal
------------------------------
The model's prediction is: Normal
```

```
------------------------------
Health Guide: Normal Weight:
------------------------------
• Maintain a balanced diet rich in fruits, vegetables, and lean proteins.
• Engage in regular physical activity, including both cardio and strength training.
• Stay hydrated and get sufficient sleep.
• Continue to monitor your health and well-being.

==================================================
Would you like to analyze another person? (yes/no):  no
```

4.2

```python
# 4.2
# Nguyen Viet Quang
import os
import json
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import accuracy_score, mean_absolute_error, mean_squared_error
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, LSTM, Conv1D
import tensorflow as tf
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline

# Suppress TensorFlow warnings
tf.get_logger().setLevel('ERROR')
```

```python
# Helper function to ensure directory exists
def create_directory(path):
    if not os.path.exists(path):
        os.makedirs(path)
        print(f"Created directory: {path}")
    else:
        print(f"Directory already exists: {path}")

# --- (a) Load the provided dataset ---
def load_dataset(file_path):
    """Loads the dataset from the specified file path."""
    print("--- (a) Loading dataset ---")
    try:
        # Assuming the file is a space-delimited text file
        df = pd.read_csv(file_path, delim_whitespace=True)
        print(f"Dataset loaded from {file_path}")
        return df
    except FileNotFoundError:
        print(f"Error: The file {file_path} was not found.")
        return None
    except Exception as e:
        print(f"An error occurred while loading the file: {e}")
        return None
```

```python
# --- (b) Show the distribution of the dataset ---
def show_distribution(df):
    """Shows the basic distribution of the dataset."""
    print("\n--- (b) Dataset Distribution ---")
    print("DataFrame Info:")
    df.info()
    print("\nDescriptive Statistics:")
    print(df.describe().round(2))
```

```python
# --- (c) Basic ML models for classification ---
def train_basic_ml_models(df):
    """Trains 5 basic ML models to classify BMI categories."""
    print("\n--- (c) Training 5 Basic ML Models ---")

    # Calculate BMI and create the target variable
    df['bmi'] = df['WEIGHT'] / (df['HEIGHT'] / 100)**2
    def classify_bmi(bmi):
        if bmi < 18.5:
            return 'Underweight'
        elif bmi >= 25.0:
            return 'Overweight'
        else:
            return 'Normal'
    df['bmi_category'] = df['bmi'].apply(classify_bmi)

    # Define features and target
    features = ['Member_Age_Orig', 'Member_Gender_Orig', 'Mod_act', 'Vig_act']
    X = df[features]
    y = df['bmi_category']

    # Preprocessing pipelines for numerical and categorical features
    numerical_features = ['Member_Age_Orig', 'Mod_act', 'Vig_act']
    numerical_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='mean')),
        ('scaler', StandardScaler())
    ])
categorical_features = ['Member_Gender_Orig']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Create the full preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ],
    remainder='passthrough'
)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit preprocessor on training data
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)

models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42),
    'Support Vector Machine (SVC)': SVC(random_state=42)
}
```

```
results = {}
for name, model in models.items():
    print(f"Training {name}...")
    model.fit(X_train_processed, y_train)
    y_pred = model.predict(X_test_processed)
    results[name] = {
        'model': model,
        'y_test': y_test,
        'y_pred': y_pred
    }

return results, preprocessor, df
```

```
# --- (d) Build CNN, RNN, LSTM models ---
def build_deep_learning_models(X_train_processed, y_train, preprocessor):
    """
    Builds and trains CNN, RNN, and LSTM models for the classification task.
    """
    print("\n--- (d) Building and Training Deep Learning Models ---")

    # Encode target labels
    label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(y_train)
    num_classes = len(label_encoder.classes_)

    # Reshape data for deep learning models
    X_train_reshaped = X_train_processed.toarray() if hasattr(X_train_processed, 'toarray') else X_train_processed
    X_train_reshaped = X_train_reshaped.reshape(X_train_reshaped.shape[0], X_train_reshaped.shape[1], 1)

    # Define the input shape
    input_shape = (X_train_reshaped.shape[1], 1)
```

```
    # --- CNN Model ---
    def build_cnn():
        model = Sequential([
            tf.keras.Input(shape=input_shape),
            Conv1D(filters=32, kernel_size=3, activation='relu'),
            Dropout(0.2),
            Conv1D(filters=64, kernel_size=3, activation='relu'),
            Dropout(0.2),
            Flatten(),
            Dense(128, activation='relu'),
            Dropout(0.3),
            Dense(num_classes, activation='softmax')
        ])
        model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
        return model

    # --- LSTM Model ---
    def build_lstm():
        model = Sequential([
            tf.keras.Input(shape=input_shape),
            LSTM(64, return_sequences=True),
            Dropout(0.2),
            LSTM(64),
            Dropout(0.2),
            Dense(64, activation='relu'),
            Dropout(0.3),
            Dense(num_classes, activation='softmax')
        ])
        model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
        return model

# --- RNN Model (using SimpleRNN) ---
def build_rnn():
    model = Sequential([
        tf.keras.Input(shape=input_shape),
        tf.keras.layers.SimpleRNN(64, return_sequences=True),
        Dropout(0.2),
        tf.keras.layers.SimpleRNN(64),
        Dropout(0.2),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

deep_models = {
    'CNN': build_cnn(),
    'LSTM': build_lstm(),
    'RNN': build_rnn()
}


    dl_results = {}
    for name, model in deep_models.items():
        print(f"\nTraining {name}...")
        model.fit(X_train_reshaped, y_encoded, epochs=10, batch_size=32, verbose=0)
        dl_results[name] = {'model': model, 'label_encoder': label_encoder}

    # Create the health guide JSON file
    create_health_guide_kb()

    return dl_results, preprocessor
```

```python
# Function to create the health guide knowledge base
def create_health_guide_kb():
    """Builds a simple JSON knowledge base for health advice."""
    health_guide = {
        "Underweight": {
            "title": "Health Guide: Underweight",
            "advice": [
                "Increase your calorie intake with nutrient-dense foods like whole grains, healthy fats, and lean proteins.",
                "Eat more frequent meals throughout the day.",
                "Incorporate strength training exercises to build muscle mass.",
                "Consult a nutritionist or doctor for a personalized plan."
            ]
        },
        "Normal": {
            "title": "Health Guide: Normal Weight",
            "advice": [
                "Maintain a balanced diet rich in fruits, vegetables, and lean proteins.",
                "Engage in regular physical activity, including both cardio and strength training.",
                "Stay hydrated and get sufficient sleep.",
                "Continue to monitor your health and well-being."
            ]
        },
        "Overweight": {
            "title": "Health Guide: Overweight",
            "advice": [
                "Focus on a diet with a calorie deficit by reducing intake of processed foods and sugary drinks.",
                "Increase daily physical activity, such as walking, jogging, or cycling.",
                "Choose smaller portion sizes and eat mindfully.",
                "Seek advice from a healthcare professional to set realistic weight loss goals."
            ]
        }
    }
    file_path = r'C:\DATA\kb_healthGuide.json'
    create_directory(os.path.dirname(file_path))
    with open(file_path, 'w') as f:
        json.dump(health_guide, f, indent=4)
    print(f"\nHealth guide knowledge base created and saved to {file_path}")

# --- (e) Compare and evaluate models ---
def evaluate_models(ml_results, dl_results, X_test, y_test, preprocessor):
    """Compares and evaluates all models with various metrics."""
    print("\n--- (e) Model Comparison and Evaluation ---")

    # Prepare data for deep learning models
    label_encoder = dl_results['CNN']['label_encoder']
    y_test_encoded = label_encoder.transform(y_test)
    X_test_processed = preprocessor.transform(X_test)
    X_test_reshaped = X_test_processed.toarray() if hasattr(X_test_processed, 'toarray') else X_test_processed
    X_test_reshaped = X_test_reshaped.reshape(X_test_reshaped.shape[0], X_test_reshaped.shape[1], 1)

    all_results = {}
```

```
# Evaluate ML models
for name, res in ml_results.items():
    y_test_num = label_encoder.transform(res['y_test'])
    y_pred_num = label_encoder.transform(res['y_pred'])
    all_results[name] = {
        'Accuracy': accuracy_score(res['y_test'], res['y_pred']),
        'MAE': mean_absolute_error(y_test_num, y_pred_num),
        'MSE': mean_squared_error(y_test_num, y_pred_num),
        'RMSE': np.sqrt(mean_squared_error(y_test_num, y_pred_num))
    }

# Evaluate Deep Learning models
for name, res in dl_results.items():
    y_pred_probs = res['model'].predict(X_test_reshaped, verbose=0)
    y_pred_encoded = np.argmax(y_pred_probs, axis=1)
    y_pred = label_encoder.inverse_transform(y_pred_encoded)

    y_pred_num = y_pred_encoded
    y_test_num = y_test_encoded

    all_results[name] = {
        'Accuracy': accuracy_score(y_test, y_pred),
        'MAE': mean_absolute_error(y_test_num, y_pred_num),
        'MSE': mean_squared_error(y_test_num, y_pred_num),
        'RMSE': np.sqrt(mean_squared_error(y_test_num, y_pred_num))
    }


# Create and display evaluation table
evaluation_df = pd.DataFrame(all_results).T.round(3)
print(evaluation_df)

# Find the best model based on accuracy
best_model_name = evaluation_df['Accuracy'].idxmax()
print(f"\nBest model based on Accuracy: {best_model_name}")

if best_model_name in ml_results:
    best_model = ml_results[best_model_name]['model']
else:
    best_model = dl_results[best_model_name]['model']

return best_model, best_model_name, preprocessor, label_encoder
```

```python
# --- (f) Visualize with >=5 various types ---
def visualize_data(df, ml_results):
    """Generates various plots to visualize the data and model results."""
    print("\n--- (f) Data Visualization ---")

    plt.style.use('seaborn-v0_8-whitegrid')
    fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(15, 18))
    fig.suptitle('Dataset and Model Evaluation Visualizations', fontsize=20, y=1.02)

    # Plot 1: Age Distribution
    sns.histplot(df['Member_Age_Orig'], bins=15, kde=True, ax=axes[0, 0], color='skyblue')
    axes[0, 0].set_title('Age Distribution')
    axes[0, 0].set_xlabel('Age (years)')
    axes[0, 0].set_ylabel('Count')

    # Plot 2: BMI Category Distribution
    sns.countplot(x='bmi_category', data=df, ax=axes[0, 1], palette='viridis', hue='bmi_category', legend=False)
    axes[0, 1].set_title('BMI Category Distribution')
    axes[0, 1].set_xlabel('BMI Category')
    axes[0, 1].set_ylabel('Count')

    # Plot 3: Scatter plot of Age vs. BMI
    sns.scatterplot(x='Member_Age_Orig', y='bmi', hue='bmi_category', data=df, ax=axes[1, 0], palette='coolwarm')
    axes[1, 0].set_title('BMI vs. Age')
    axes[1, 0].set_xlabel('Age (years)')
    axes[1, 0].set_ylabel('BMI')
```

```python
# Plot 4: Gender Distribution
sns.countplot(y='Member_Gender_Orig', data=df, ax=axes[1, 1], palette='plasma', order=df['Member_Gender_Orig'].value_counts().index, hue='Member_Gen
axes[1, 1].set_title('Gender Distribution')
axes[1, 1].set_xlabel('Count')
axes[1, 1].set_ylabel('Gender')

# Plot 5: Height vs. Weight Scatter Plot with BMI Categories
sns.scatterplot(x='HEIGHT', y='WEIGHT', hue='bmi_category', data=df, ax=axes[2, 0], palette='magma')
axes[2, 0].set_title('Height vs. Weight')
axes[2, 0].set_xlabel('Height (cm)')
axes[2, 0].set_ylabel('Weight (kg)')

# Plot 6: Confusion Matrix for a sample model (e.g., Random Forest)
best_ml_model_name = max(ml_results, key=lambda k: accuracy_score(ml_results[k]['y_test'], ml_results[k]['y_pred']))
y_pred = ml_results[best_ml_model_name]['y_pred']
y_test = ml_results[best_ml_model_name]['y_test']

confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
sns.heatmap(confusion_matrix, annot=True, fmt='d', cmap='Blues', ax=axes[2, 1])
axes[2, 1].set_title(f'Confusion Matrix for {best_ml_model_name}')

plt.tight_layout(pad=3.0)
plt.show()
```

```python
# --- (g) Deploy the best model ---
def deploy_model(best_model, best_model_name, preprocessor, label_encoder):
    """
    Deploys the best model, takes user input, makes a prediction,
    and provides a health guide.
    """
    print(f"\n" + "="*50)
    print(f"--- (g) Deploying the Best Model: {best_model_name} ---")
    print("="*50)

    # Load the health guide knowledge base
    try:
        with open(r'C:\DATA\kb_healthGuide.json', 'r') as f:
            health_guide_kb = json.load(f)
    except FileNotFoundError:
        print("Health guide file not found. Please run the full script to generate it.")
        return

    def get_user_input():
        """Prompts the user for input with validation."""
        print("\nPlease enter your details to get a health analysis.")
```

```python
while True:
    try:
        age = int(input("Enter your age (20-60 years): "))
        if not (20 <= age <= 60):
            print("Age must be between 20 and 60. Please try again.")
            continue
        break
    except ValueError:
        print("Invalid input. Please enter a number for age.")

genders = ['Male', 'Female', 'Other']
print(f"Available genders: {', '.join(genders)}")
while True:
    gender = input("Enter your gender: ").strip().title()
    if gender not in genders:
        print("Invalid gender. Please choose from the list.")
        continue
    break

while True:
    try:
        height = float(input("Enter your height in cm: "))
        if height <= 0:
            print("Height must be a positive number. Please try again.")
            continue
        break
    except ValueError:
        print("Invalid input. Please enter a number for height.")
```

```
while True:
    try:
        weight = float(input("Enter your weight in kg: "))
        if weight <= 0:
            print("Weight must be a positive number. Please try again.")
            continue
        break
    except ValueError:
        print("Invalid input. Please enter a number for weight.")

while True:
    try:
        mod_act = float(input("Enter your hours of moderate activity per week: "))
        if mod_act < 0:
            print("Activity hours cannot be negative. Please try again.")
            continue
        break
    except ValueError:
        print("Invalid input. Please enter a number for moderate activity.")

while True:
    try:
        vig_act = float(input("Enter your hours of vigorous activity per week: "))
        if vig_act < 0:
            print("Activity hours cannot be negative. Please try again.")
            continue
        break
    except ValueError:
        print("Invalid input. Please enter a number for vigorous activity.")

return age, gender, height, weight, mod_act, vig_act

def predict_and_advise(age, gender, height, weight, mod_act, vig_act):
    """
    Takes user input, makes a BMI prediction, and provides health advice.
    """
    # Calculate BMI
    bmi = weight / (height / 100)**2

    # Preprocess user input
    user_data = pd.DataFrame([[age, gender, mod_act, vig_act]],
                             columns=['Member_Age_Orig', 'Member_Gender_Orig', 'Mod_act', 'Vig_act'])
    user_data_processed = preprocessor.transform(user_data)

    # Make prediction
    if best_model_name in ['CNN', 'RNN', 'LSTM']:
        # Convert to dense array and then reshape for deep learning models
        user_data_processed = user_data_processed.toarray()
        user_data_processed = user_data_processed.reshape(1, user_data_processed.shape[1], 1)
        prediction_probs = best_model.predict(user_data_processed, verbose=0)
        prediction_encoded = np.argmax(prediction_probs, axis=1)
        predicted_category = label_encoder.inverse_transform(prediction_encoded)[0]
    else:
        predicted_category = best_model.predict(user_data_processed)[0]
```

```
    # Get the actual BMI category
    def classify_bmi(bmi):
        if bmi < 18.5:
            return 'Underweight'
        elif bmi >= 25.0:
            return 'Overweight'
        else:
            return 'Normal'
    actual_category = classify_bmi(bmi)

    print("\n" + "-"*30)
    print("--- Your Health Analysis ---")
    print("-"*30)
    print(f"Based on your inputs:")
    print(f"  Age: {age} years")
    print(f"  Gender: {gender}")
    print(f"  Height: {height} cm")
    print(f"  Weight: {weight} kg")
    print(f"  Moderate Activity: {mod_act} hours/week")
    print(f"  Vigorous Activity: {vig_act} hours/week")
    print("\n" + "-"*30)
    print(f"Your calculated BMI is: {bmi:.2f}")
    print(f"This corresponds to the category: {actual_category}")
    print("-" * 30)

    # Display the predicted category and health guide
    print(f"The model's prediction is: {predicted_category}")

    advice = health_guide_kb.get(predicted_category, {"title": "No Advice Found", "advice": []})
    print(f"\n" + "-"*30)
    print(f"{advice['title']}:")
    print("-" * 30)
    for item in advice['advice']:
        print(f"• {item}")
    print("\n" + "="*50)

# Loop for user input
while True:
    age_input, gender_input, height_input, weight_input, mod_act_input, vig_act_input = get_user_input()
    predict_and_advise(age=age_input, gender=gender_input, height=height_input, weight=weight_input, mod_act=mod_act_input, vig_act=vig_act_input)

    another = input("Would you like to analyze another person? (yes/no): ").lower()
    if another != 'yes':
        break
```

```python
# Main script execution block
if __name__ == "__main__":
    # a. Load dataset
    file_path = r'D:\school\se1_year4\IntSys\assignment4\data.txt'
    df = load_dataset(file_path)

    if df is not None:
        # b. Show dataset distribution
        show_distribution(df)

        # c. Train basic ML models
        ml_results, preprocessor, df_with_bmi = train_basic_ml_models(df)

        # d. Build and train deep learning models & create health guide KB
        X_train_processed = preprocessor.fit_transform(df_with_bmi[['Member_Age_Orig', 'Member_Gender_Orig', 'Mod_act', 'Vig_act']])
        y_train = df_with_bmi['bmi_category']
        dl_results, _ = build_deep_learning_models(X_train_processed, y_train, preprocessor)

        # e. Compare and evaluate all models
        X_test = df_with_bmi[['Member_Age_Orig', 'Member_Gender_Orig', 'Mod_act', 'Vig_act']].loc[ml_results['Random Forest']['y_test'].index]
        y_test = ml_results['Random Forest']['y_test']
        best_model, best_model_name, preprocessor, label_encoder = evaluate_models(
            ml_results, dl_results, X_test, y_test, preprocessor
        )

        # f. Visualize
        visualize_data(df_with_bmi, ml_results)
```

```python
# f. Visualize
visualize_data(df_with_bmi, ml_results)

# g. Deploy the best model
deploy_model(best_model, best_model_name, preprocessor, label_encoder)
```

output

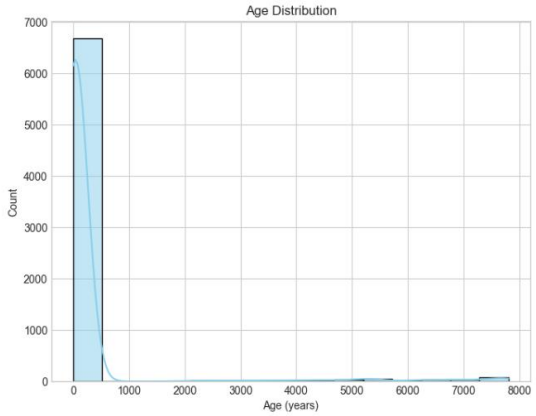
```
--- (a) Loading dataset ---
Dataset loaded from D:\school\se1_year4\IntSys\assignment4\data.txt

--- (b) Dataset Distribution ---
DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7072 entries, 0 to 7071
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   HH_ID               7072 non-null   int64
 1   Member_Age_Orig     7072 non-null   float64
 2   Member_ID           7049 non-null   float64
 3   Member_Gender_Orig  6982 non-null   float64
 4   HEIGHT              6484 non-null   float64
 5   WEIGHT              6358 non-null   float64
 6   Mod_act             5882 non-null   float64
 7   Vig_act             3971 non-null   float64
dtypes: float64(7), int64(1)
memory usage: 442.1 KB
```

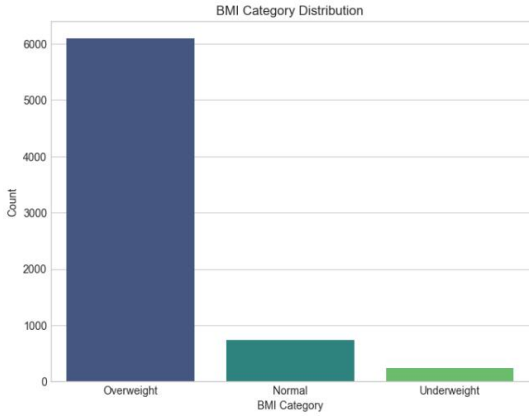
```
Descriptive Statistics:
          HH_ID  Member_Age_Orig  Member_ID  Member_Gender_Orig   HEIGHT  \
count   7072.00          7072.00    7049.00             6982.00  6484.00
mean    7329.38           338.47    3643.00                2.33    11.75
std     4275.08          1294.54    2359.69               10.31    30.19
min        1.00             0.00       1.00                1.00     0.00
25%     3708.00            23.00    1552.00                1.00     5.11
50%     6999.00            48.00    3582.00                2.00     5.50
75%    10951.75            64.00    5713.00                2.00     5.90
max    14998.00          7805.00    7798.00              285.00   250.00

         WEIGHT  Mod_act  Vig_act
count  6358.00  5882.00  3971.00
mean    145.27    13.49     4.72
std      65.96    14.32     9.94
min       0.00     0.00     0.00
25%     120.00     5.00     0.00
50%     155.00     9.00     2.00
75%     185.00    20.00     5.00
max     500.00   150.00   275.00
```

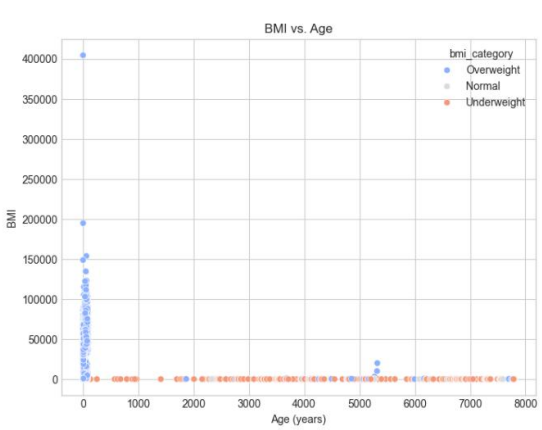
```
--- (c) Training 5 Basic ML Models ---
Training Logistic Regression...
Training Decision Tree...
Training Random Forest...
Training Gradient Boosting...
Training Support Vector Machine (SVC)...

--- (d) Building and Training Deep Learning Models ---

Training CNN...

Training LSTM...

Training RNN...
Directory already exists: C:\DATA

Health guide knowledge base created and saved to C:\DATA\kb_healthGuide.json
```

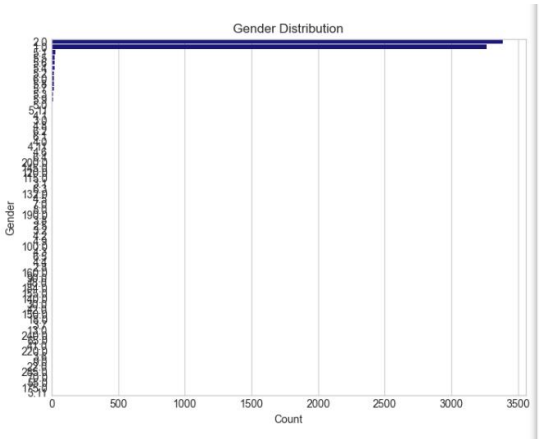
```
--- (e) Model Comparison and Evaluation ---
                              Accuracy    MAE    MSE   RMSE
Logistic Regression              0.907  0.100  0.112  0.335
Decision Tree                    0.927  0.086  0.111  0.333
Random Forest                    0.933  0.075  0.090  0.301
Gradient Boosting                0.937  0.071  0.086  0.294
Support Vector Machine (SVC)     0.908  0.099  0.113  0.336
CNN                              0.938  0.069  0.081  0.285
LSTM                             0.875  0.125  0.125  0.354
RNN                              0.923  0.087  0.107  0.327

Best model based on Accuracy: CNN
```

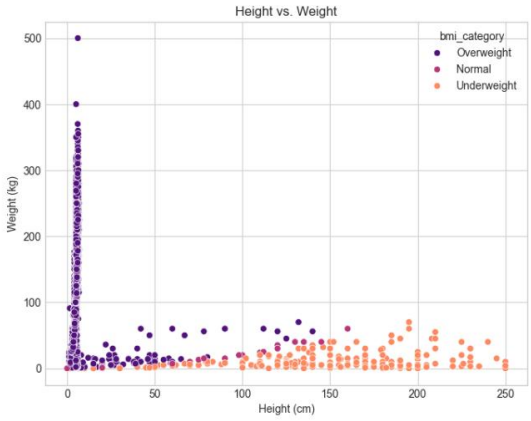
Dataset and Model Evaluation Visualizations











```
==================================================
--- (g) Deploying the Best Model: CNN ---
==================================================

Please enter your details to get a health analysis.
Enter your age (20-60 years):  20
Available genders: Male, Female, Other
Enter your gender:  Female
Enter your height in cm:  160
Enter your weight in kg:  55
Enter your hours of moderate activity per week:  5
Enter your hours of vigorous activity per week:  3


------------------------------
--- Your Health Analysis ---
------------------------------
Based on your inputs:
  Age: 20 years
  Gender: Female
  Height: 160.0 cm
  Weight: 55.0 kg
  Moderate Activity: 5.0 hours/week
  Vigorous Activity: 3.0 hours/week

------------------------------
Your calculated BMI is: 21.48
This corresponds to the category: Normal
------------------------------
The model's prediction is: Overweight

------------------------------
Health Guide: Overweight:
------------------------------
• Focus on a diet with a calorie deficit by reducing intake of processed foods and sugary drinks.
• Increase daily physical activity, such as walking, jogging, or cycling.
• Choose smaller portion sizes and eat mindfully.
• Seek advice from a healthcare professional to set realistic weight loss goals.

==================================================
Would you like to analyze another person? (yes/no):  no
```