Import

```
import random
import numpy as np
import pandas as pd
from collections import Counter
import matplotlib.pyplot as plt
import seaborn as sns
import joblib

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
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

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (Dense, Dropout, Flatten, Conv1D,
                                     LSTM, SimpleRNN, Input, BatchNormalization)
from tensorflow.keras.utils import to_categorical
```

Helper utilities

```python
# ----------------------------
# Helper utilities
# ----------------------------
def create_directory(path):
    """Creates a directory if it doesn't exist."""
    os.makedirs(path, exist_ok=True)

def save_dataframe_csv(df, path):
    """Saves a DataFrame to a CSV file."""
    create_directory(os.path.dirname(path))
    df.to_csv(path, index=False)
    print(f"Saved CSV -> {path}")

def write_json(obj, path):
    """Saves a Python object to a JSON file."""
    create_directory(os.path.dirname(path))
    with open(path, 'w', encoding='utf-8') as f:
        json.dump(obj, f, ensure_ascii=False, indent=2)
    print(f"Saved JSON -> {path}")

def save_joblib(obj, path):
    """Saves a Python object using joblib."""
    create_directory(os.path.dirname(path))
    joblib.dump(obj, path)
    print(f"Saved joblib object -> {path}")
```

a. Build Dataset

Sinh viên xây dựng dataset với 10.000 người, gồm 20 features như: age, gender, job, height, weight, area, diet, exercise, sleep_hours, smoking, alcohol, stress_level, blood_pressure, cholesterol, glucose, family_history, education, income, marital_status, hobbies. Dataset được lưu tại C:\DATA\data_4.2.csv.

```python
# ---------------------------
# (a) Build dataset (10000 x 20)
# ---------------------------
def generate_dataset(num_persons=10000, seed=42):
    """Generates a synthetic health dataset."""
    np.random.seed(seed)
    random.seed(seed)
    jobs = ['Engineer', 'Doctor', 'Teacher', 'Artist', 'Programmer', 'Lawyer',
            'Accountant', 'Student', 'Scientist', 'Nurse', 'Driver', 'Chef', 'Farmer']
    genders = ['Male', 'Female', 'Other']
    areas = ['Urban', 'Suburban', 'Rural']
    diets = ['Omnivore', 'Vegetarian', 'Vegan', 'Keto', 'Mediterranean', 'Pescatarian']
    data = {
        'age': np.random.randint(18, 80, num_persons),
        'job': np.random.choice(jobs, num_persons),
        'gender': np.random.choice(genders, num_persons, p=[0.49, 0.49, 0.02]),
        'area': np.random.choice(areas, num_persons, p=[0.5, 0.3, 0.2]),
        'diet_type': np.random.choice(diets, num_persons),
        'height': np.random.normal(168, 10, num_persons).round(1),
        'weight': np.random.normal(72, 16, num_persons).round(1),
        'smoking': np.random.choice([0,1], num_persons, p=[0.8, 0.2]),
        'alcohol': np.random.poisson(2, num_persons),
        'exercise_hours': np.random.normal(3.5, 2.5, num_persons).round(1).clip(0, 20),
        'sleep_hours': np.random.normal(7, 1.5, num_persons).round(1).clip(3, 12),
        'blood_pressure_sys': np.random.normal(122, 15, num_persons).round(0),
        'blood_pressure_dia': np.random.normal(78, 10, num_persons).round(0),
        'cholesterol': np.random.normal(195, 40, num_persons).round(0),
        'glucose': np.random.normal(98, 20, num_persons).round(0),
        'steps_per_day': np.random.normal(6000, 3000, num_persons).round(0).clip(0, 30000),
        'calories_intake': np.random.normal(2200, 500, num_persons).round(0).clip(1000, 5000),
        'protein_intake': np.random.normal(80, 25, num_persons).round(0).clip(10, 300),
        'screen_time': np.random.normal(5, 3, num_persons).round(1).clip(0, 18),
        'stress_level': np.random.randint(1, 11, num_persons)
    }
    df = pd.DataFrame(data)
    df['height'] = df['height'].apply(lambda x: float(max(140.0, min(x, 210.0))))
    df['weight'] = df['weight'].apply(lambda x: float(max(35.0, min(x, 200.0))))
    df['bmi'] = (df['weight'] / ((df['height'] / 100) ** 2)).round(1)
    def bmi_to_label(bmi):
        if bmi < 18.5: return 'Underweight'
        elif bmi < 25.0: return 'Normal'
        else: return 'Overweight'
    df['bmi_category'] = df['bmi'].apply(bmi_to_label)

    # Save to file
    save_dataframe_csv(df, r'C:\DATA\data_4.2.csv')

    return df
```

b. Show Dataset Distribution

Hiển thị thống kê mô tả, phân bố độ tuổi, giới tính, nghề nghiệp, BMI.

```python
# ---------------------------
# (b) Show the distribution of the dataset
# ---------------------------
def show_distribution(df):
    """
    Prints the distribution of the dataset and saves plots.
    """
    print("\n--- Dataset Distribution ---")
    print(df['bmi_category'].value_counts())
    print("\nGender distribution:\n", df['gender'].value_counts())
    print("\nJob distribution:\n", df['job'].value_counts())
    print("\nArea distribution:\n", df['area'].value_counts())
    print("\nDiet distribution:\n", df['diet_type'].value_counts())
```

Output

```
--- Dataset Distribution ---
bmi_category
Overweight     5390
Normal         3379
Underweight    1231
Name: count, dtype: int64

Gender distribution:
 gender
Female    4932
Male      4837
Other      231
Name: count, dtype: int64
```

```
Job distribution:
 job
Nurse          810
Farmer         802
Accountant     782
Chef           779
Lawyer         778
Scientist      772
Programmer     768
Engineer       768
Doctor         764
Teacher        757
Artist         754
Student        744
Driver         722
Name: count, dtype: int64

Area distribution:
 area
Urban       4985
Suburban    3059
Rural       1956
Name: count, dtype: int64

Diet distribution:
 diet_type
Keto             1741
Vegetarian       1727
Pescatarian      1674
Mediterranean    1634
Vegan            1615
Omnivore         1609
Name: count, dtype: int64
```

c. Basic ML Models

Sử dụng 5 mô hình cơ bản Logistic Regression, Decision Tree, Random Forest, Gradient Boosting, và SVM. Mục tiêu: phân loại tình trạng BMI thành Underweight, Normal, Overweight.

```
# ---------------------------
# (c) Train 5 basic ML models
# ---------------------------
def train_basic_ml_models(df):
    """Trains and evaluates 5 traditional ML models."""
    print("\n--- Training 5 basic ML models ---")
    feature_cols = [col for col in df.columns if col not in ['bmi', 'bmi_category']]
    X = df[feature_cols].copy()
    y = df['bmi_category'].copy()
    categorical_features = ['job', 'gender', 'area', 'diet_type']
    numeric_features = [c for c in feature_cols if c not in categorical_features]
    try:
        ohe = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
    except TypeError:
        ohe = OneHotEncoder(handle_unknown='ignore', sparse=False)
    preprocessor = ColumnTransformer(transformers=[
        ('cat', ohe, categorical_features),
        ('num', StandardScaler(), numeric_features)
    ], remainder='drop')
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
    models = {
        'Logistic Regression': LogisticRegression(max_iter=1500),
        'Decision Tree': DecisionTreeClassifier(random_state=42),
        'Random Forest': RandomForestClassifier(n_estimators=150, random_state=42),
        'Gradient Boosting': GradientBoostingClassifier(n_estimators=150, random_state=42),
        'SVC': SVC(probability=True, random_state=42)
    }
```

```
    trained = {}
    for name, mdl in models.items():
        print(f"Training {name} ...")
        pipe = Pipeline([('pre', preprocessor), ('clf', mdl)])
        pipe.fit(X_train, y_train)
        y_pred = pipe.predict(X_test)
        trained[name] = {
            'pipeline': pipe,
            'y_test': y_test.reset_index(drop=True),
            'y_pred': pd.Series(y_pred)
        }
        acc = accuracy_score(y_test, y_pred)
        print(f"  {name} accuracy: {acc:.4f}")
    return trained, preprocessor, X_test, y_test
```

Output

```
--- Training 5 basic ML models ---
Training Logistic Regression ...
  Logistic Regression accuracy: 0.9885
Training Decision Tree ...
  Decision Tree accuracy: 0.9740
Training Random Forest ...
  Random Forest accuracy: 0.9490
Training Gradient Boosting ...
  Gradient Boosting accuracy: 0.9785
Training SVC ...
  SVC accuracy: 0.9570
```

d. CNN, RNN, LSTM Models

Xây dựng mô hình CNN, RNN, LSTM với >= 5 layers. Dữ liệu tabular được reshape thành dạng sequence để phù hợp với Conv1D và RNN.

```python
# ---------------------------
# (d) Build DL models (CNN, RNN, LSTM)
# ---------------------------
def build_and_train_deep_models(df, preprocessor, epochs=8):
    """Builds and trains deep learning models."""
    print("\n--- Building & training deep learning models ---")
    feature_cols = [col for col in df.columns if col not in ['bmi', 'bmi_category']]
    X_all = df[feature_cols]
    y_all = df['bmi_category']
    X_trans = preprocessor.fit_transform(X_all)
    if hasattr(X_trans, "toarray"):
        X_trans = X_trans.toarray()
    le = LabelEncoder()
    y_enc = le.fit_transform(y_all)
    num_classes = len(le.classes_)
    X_tr, X_te, y_tr, y_te = train_test_split(X_trans, y_enc, test_size=0.2, random_state=42, stratify=y_enc)
    X_tr_r = X_tr.reshape(X_tr.shape[0], X_tr.shape[1], 1)
    X_te_r = X_te.reshape(X_te.shape[0], X_te.shape[1], 1)
    input_shape = (X_tr_r.shape[1], 1)
```

```
def build_cnn():
    model = Sequential([
        Input(shape=input_shape),
        Conv1D(64, kernel_size=3, activation='relu', padding='same'),
        BatchNormalization(),
        Conv1D(64, kernel_size=3, activation='relu', padding='same'),
        Dropout(0.25),
        Conv1D(128, kernel_size=3, activation='relu', padding='same'),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.4),
        Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

```
def build_lstm():
    model = Sequential([
        Input(shape=input_shape),
        LSTM(128, return_sequences=True),
        Dropout(0.25),
        LSTM(64, return_sequences=True),
        Dropout(0.2),
        LSTM(32),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

```
def build_rnn():
    model = Sequential([
        Input(shape=input_shape),
        SimpleRNN(128, return_sequences=True),
        Dropout(0.25),
        SimpleRNN(64, return_sequences=True),
        Dropout(0.2),
        SimpleRNN(32),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
models = {
    'CNN': build_cnn(),
    'LSTM': build_lstm(),
    'RNN': build_rnn()
}
trained_dl = {}
```

```python
for name, model in models.items():
    print(f"\nTraining DL model: {name} ...")
    model.fit(X_tr_r, y_tr, validation_data=(X_te_r, y_te), epochs=epochs, batch_size=64, verbose=0)
    trained_dl[name] = {
        'model': model,
        'label_encoder': le,
        'preprocessor': preprocessor,
        'X_test_reshaped': X_te_r,
        'y_test': y_te
    }
return trained_dl
```

```
--- Building & training deep learning models ---

Training DL model: CNN ...

Training DL model: LSTM ...

Training DL model: RNN ...

--- Evaluating all models ---
```

e. Compare & Evaluate Models

Đánh giá mô hình bằng các metrics: Accuracy, MAE, MSE, RMSE. Chọn mô hình tốt nhất dựa trên Accuracy.

```python
# (e) Compare and evaluate models
# ---------------------------
def evaluate_all_models(ml_results, dl_results, preprocessor, X_test_ml, y_test_ml):
    """
    Evaluates all trained models and returns the best one.
    """
    print("\n--- Evaluating all models ---")
    any_dl = next(iter(dl_results.values()))
    le = any_dl['label_encoder']
    all_metrics = {}
    for name, res in ml_results.items():
        y_test = res['y_test']
        y_pred = res['y_pred']
        y_test_num = le.transform(y_test)
        y_pred_num = le.transform(y_pred)
        acc = 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(mse)
        all_metrics[name] = {'Accuracy': acc, 'MAE': mae, 'MSE': mse, 'RMSE': rmse}
    for name, info in dl_results.items():
        model = info['model']
        X_te_r = info['X_test_reshaped']
        y_te = info['y_test']
        y_pred_probs = model.predict(X_te_r, verbose=0)
        y_pred_num = np.argmax(y_pred_probs, axis=1)
        acc = accuracy_score(y_te, y_pred_num)
        mae = mean_absolute_error(y_te, y_pred_num)
        mse = mean_squared_error(y_te, y_pred_num)

        acc = accuracy_score(y_te, y_pred_num)
        mae = mean_absolute_error(y_te, y_pred_num)
        mse = mean_squared_error(y_te, y_pred_num)
        rmse = np.sqrt(mse)
        all_metrics[name] = {'Accuracy': acc, 'MAE': mae, 'MSE': mse, 'RMSE': rmse}
    eval_df = pd.DataFrame(all_metrics).T.round(4)
    print("\nEvaluation table:\n", eval_df)
    best_model_name = eval_df['Accuracy'].idxmax()
    print(f"\nBest model by Accuracy: {best_model_name}")
    return eval_df, best_model_name, ml_results, dl_results
```

Evaluation table:

	Accuracy	MAE	MSE	RMSE
Logistic Regression	0.9885	0.0150	0.0220	0.1483
Decision Tree	0.9740	0.0335	0.0485	0.2202
Random Forest	0.9490	0.0720	0.1140	0.3376
Gradient Boosting	0.9785	0.0295	0.0455	0.2133
SVC	0.9570	0.0600	0.0940	0.3066
CNN	0.9700	0.0455	0.0765	0.2766
LSTM	0.7520	0.2730	0.3230	0.5683
RNN	0.9075	0.1395	0.2335	0.4832

f. Visualization

- Tuổi nào có nhiều người thừa cân?
- Nghề nào có nhiều người thừa cân?
- Giới tính nào có nhiều người thừa cân?
- Khu vực nào có nhiều người thừa cân?

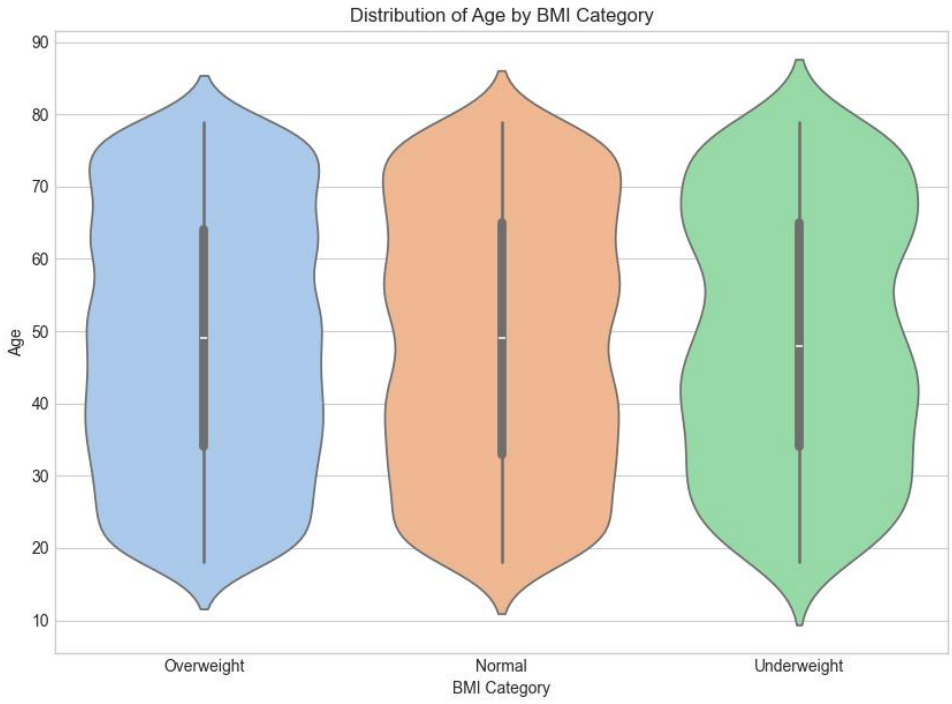
Trực quan hóa bằng histogram, barplot, scatterplot, heatmap.

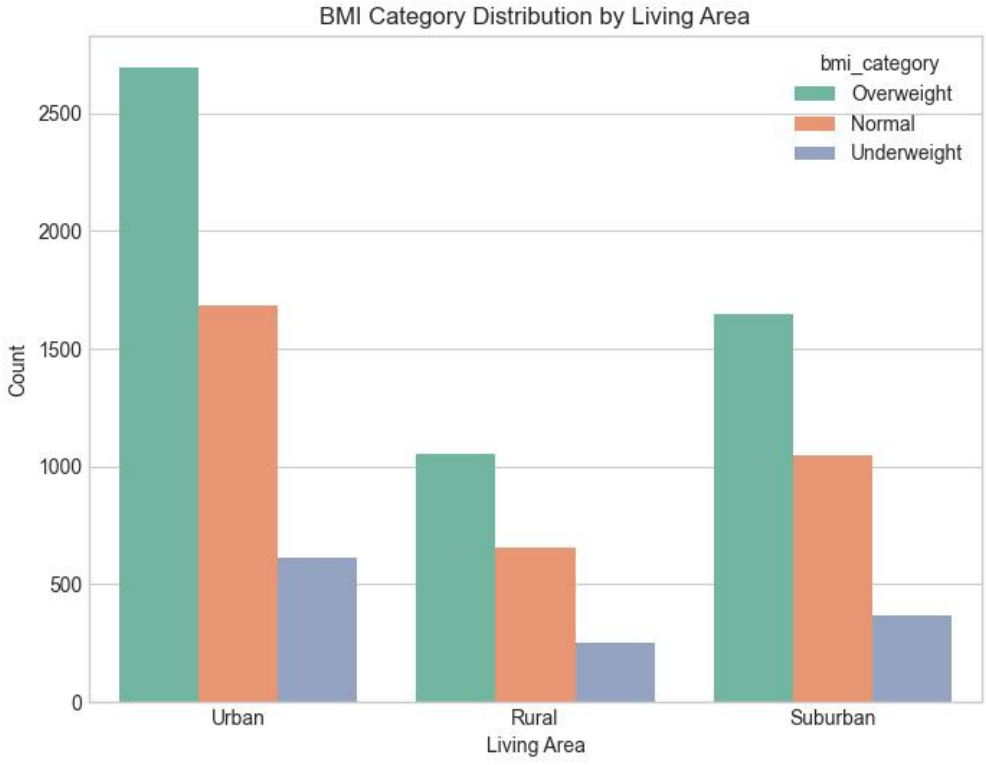
```
# ----------------------------
# (f) Visualize the results
# ----------------------------
def visualize_results(df):
    """Generates and saves various plots about the dataset."""
    print("\n--- Visualizing key relationships ---")
    create_directory(r'C:\DATA\plots/')
    plt.style.use('seaborn-v0_8-whitegrid')

    # Age vs BMI Category
    plt.figure(figsize=(10, 7))
    # Fix FutureWarning: Pass `x` as `hue` and set `legend=False`
    sns.violinplot(x='bmi_category', y='age', data=df, palette='pastel', hue='bmi_category', legend=False)
    plt.title('Distribution of Age by BMI Category')
    plt.xlabel('BMI Category')
    plt.ylabel('Age')
    plt.savefig(r'C:\DATA\plots/age_by_bmi_category.png')
    plt.close()
```

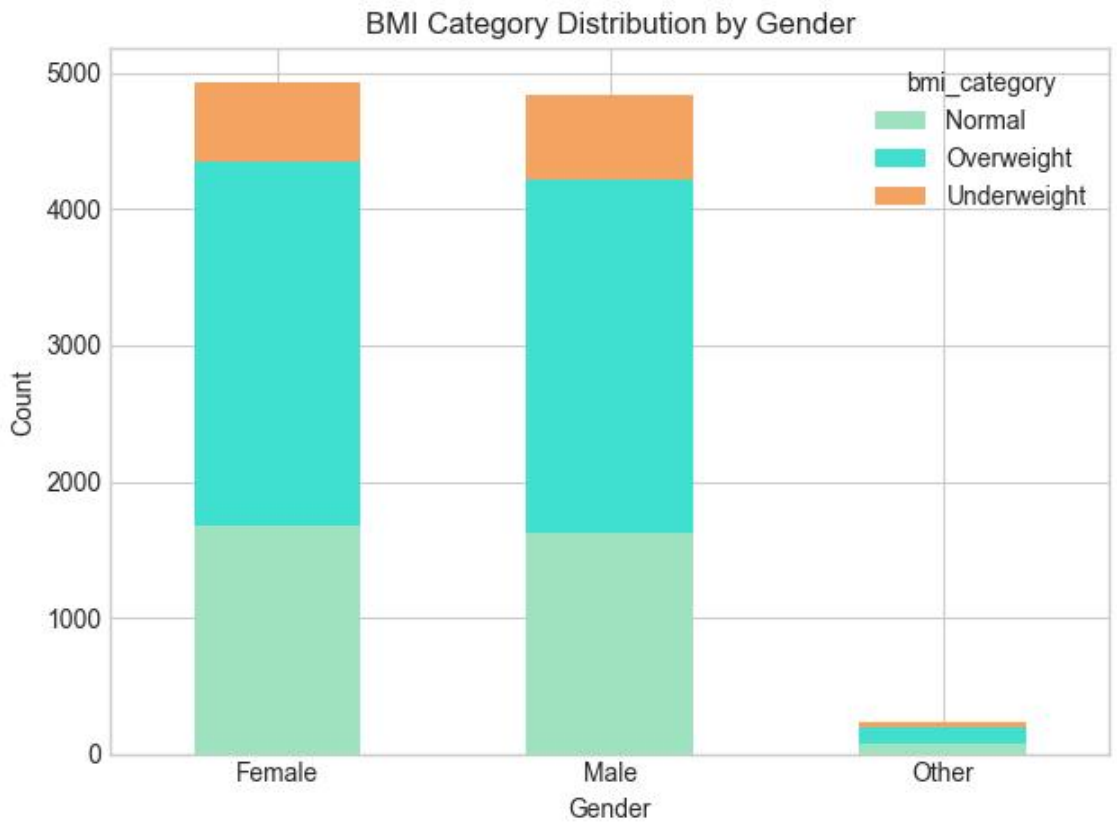
```
# Job vs BMI Category
plt.figure(figsize=(12, 8))
job_counts = df.groupby(['job', 'bmi_category']).size().unstack(fill_value=0)
job_counts['Total'] = job_counts.sum(axis=1)
job_overweight_pct = (job_counts['Overweight'] / job_counts['Total']).sort_values(ascending=False)
# Fix FutureWarning: Pass `x` as `hue` and set `legend=False`
sns.barplot(x=job_overweight_pct.index, y=job_overweight_pct.values, palette='coolwarm', hue=job_overweight_pct.index, legend=False)
plt.title('Percentage of Overweight Persons by Job')
plt.xlabel('Job')
plt.ylabel('Percentage Overweight')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.savefig(r'C:\DATA\plots/job_overweight_percentage.png')
plt.close()

# Gender vs BMI Category
plt.figure(figsize=(8, 6))
gender_counts = df.groupby(['gender', 'bmi_category']).size().unstack(fill_value=0)
gender_counts.plot(kind='bar', stacked=True, color=['#9fe2bf', '#40e0d0', '#f4a460'])
plt.title('BMI Category Distribution by Gender')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.tight_layout()
plt.savefig(r'C:\DATA\plots/gender_bmi_stacked.png')
plt.close()
```

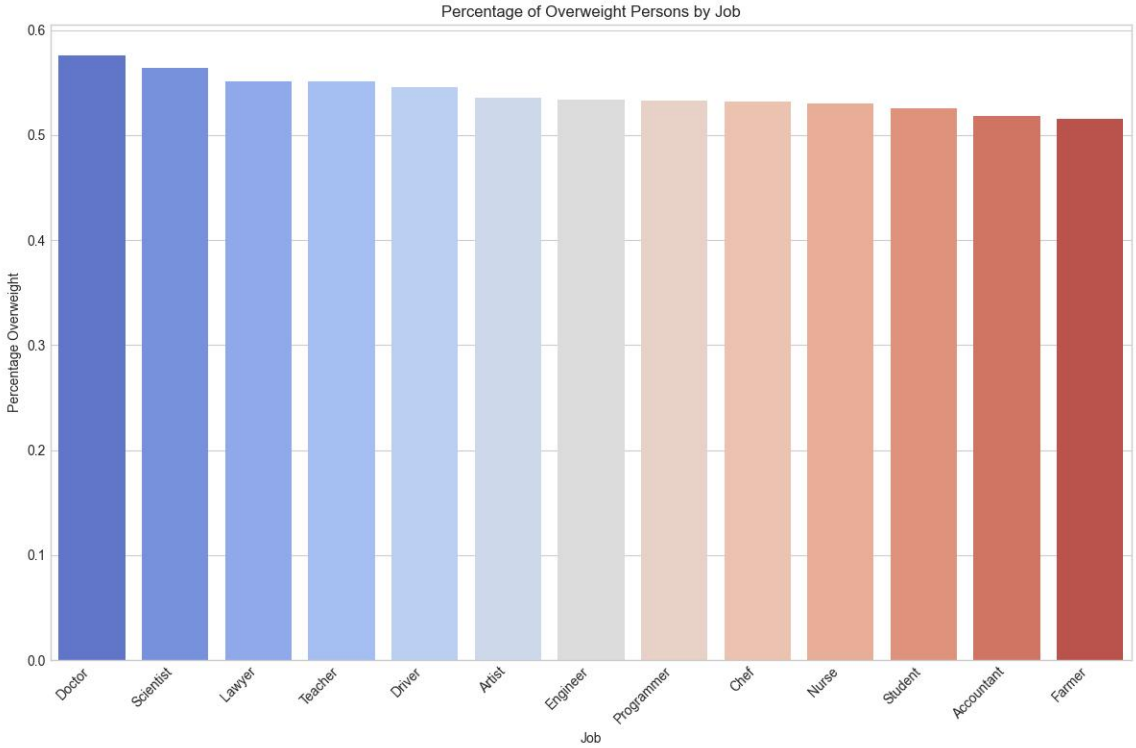
```
# Area vs BMI Category
plt.figure(figsize=(8, 6))
sns.countplot(x='area', hue='bmi_category', data=df, palette='Set2')
plt.title('BMI Category Distribution by Living Area')
plt.xlabel('Living Area')
plt.ylabel('Count')
plt.savefig(r'C:\DATA\plots/area_bmi_category.png')
plt.close()

print("Visualizations saved to 'C:\DATA\plots/' directory.")
```

Output


Distribution of Age by BMI Category
Age
90
80
70
60
50
40
30
20
10
Overweight
Normal
Underweight
BMI Category


BMI Category Distribution by Living Area
bmi_category
Overweight
Normal
Underweight
Count
2500
2000
1500
1000
500
0
Urban
Rural
Suburban
Living Area


BMI Category Distribution by Gender
bmi_category
Normal
Overweight
Underweight
Count
5000
4000
3000
2000
1000
0
Female
Male
Other
Gender

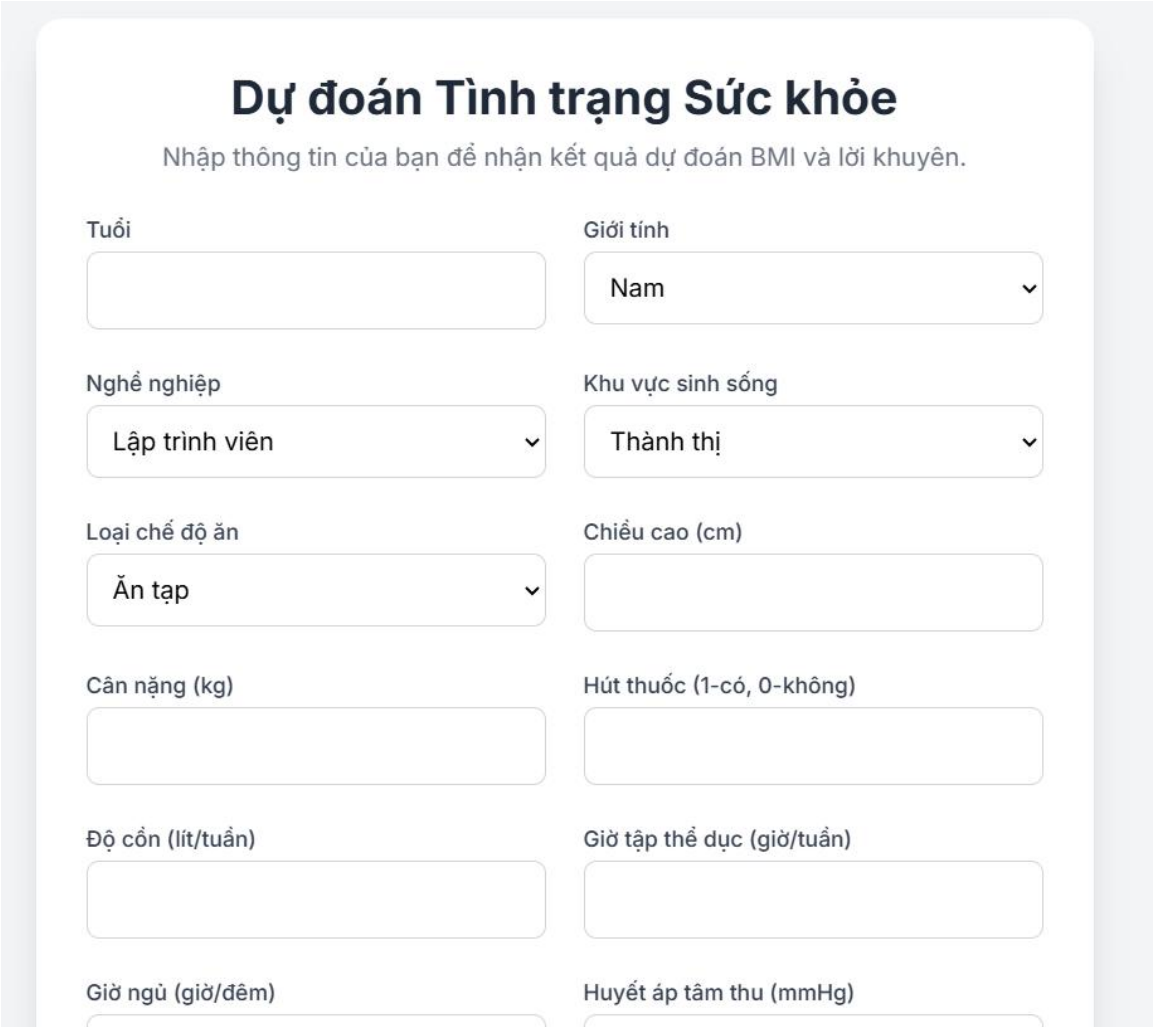

Percentage of Overweight Persons by Job
Percentage Overweight
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Doctor
Scientist
Lawyer
Teacher
Driver
Artist
Engineer
Programmer
Chef
Nurse
Student
Accountant
Farmer
Job

g. Deploy Best Model

Triển khai mô hình tốt nhất. Người dùng nhập thông tin (age, job, gender, area, diet...) → Output: Underweight, Normal, Overweight + Health Advice.

```
# 2. Tạo cơ sở dữ liệu kiến thức (KB)
kb = create_big_kb()

# 3. Lưu tất cả các thành phần cần thiết
create_directory('models/')

# Lưu tên mô hình tốt nhất
with open('models/best_model.txt', 'w') as f:
    f.write(best_name)

# Lưu bộ tiền xử lý và mô hình tốt nhất
if best_name in ml_results:
    print(f"Lưu mô hình ML tốt nhất: {best_name}")
    best_pipe = ml_results[best_name]['pipeline']
    save_joblib(best_pipe, f"models/ml_pipeline.joblib")
    # Lưu preprocessor và label encoder riêng cho DL (nếu cần)
    dl_le = dl_results[next(iter(dl_results))]['label_encoder']
    save_joblib(preprocessor_ml, f"models/dl_preprocessor.joblib")
    save_joblib(dl_le, f"models/dl_label_encoder.joblib")
else:
    print(f"Lưu mô hình DL tốt nhất: {best_name}")
    best_model_dl = dl_results[best_name]['model']
    best_model_dl.save(f"models/dl_model") # TensorFlow format
    dl_le = dl_results[best_name]['label_encoder']
    save_joblib(dl_le, f"models/dl_label_encoder.joblib")
    # Lưu preprocessor ML riêng (nếu cần)
    save_joblib(preprocessor_ml, f"models/ml_preprocessor.joblib")

 else:
     print(f"Lưu mô hình DL tốt nhất: {best_name}")
     best_model_dl = dl_results[best_name]['model']
     best_model_dl.save(f"models/dl_model") # TensorFlow format
     dl_le = dl_results[best_name]['label_encoder']
     save_joblib(dl_le, f"models/dl_label_encoder.joblib")
     # Lưu preprocessor ML riêng (nếu cần)
     save_joblib(preprocessor_ml, f"models/ml_preprocessor.joblib")

 # Lưu cơ sở dữ liệu kiến thức
 write_json(kb, f"models/kb.json")
```

Output



h. Build Knowledge Base

Xây dựng Knowledge Base >3MB lưu ở C:\DATA\kb_healthGuide.json. KB chứa lời khuyên sức khỏe chi tiết cho từng nhóm.

```python
# ---------------------------
# (h) Build BIG knowledge base >3MB
# ---------------------------
def create_big_kb(target_bytes=3_200_000):
    """
    Generates a large knowledge base and saves it to a JSON file.
    """
    print("\n--- Generating big KB (this may take a few seconds) ---")
    base_kb = {
        "Underweight": {
            "title": "Health Guide: Underweight",
            "advice": ["Increase calorie intake focusing on nutrient-dense foods (nuts, dairy, lean meats).", "Eat frequent meals and include healthy sn
        },
        "Normal": {
            "title": "Health Guide: Normal Weight",
            "advice": ["Maintain a balanced diet rich in vegetables, fruits and lean proteins.", "Keep up regular physical activity—mix cardio and stren
        },
        "Overweight": {
            "title": "Health Guide: Overweight",
            "advice": ["Aim for a modest calorie deficit; reduce sugary and processed foods.", "Increase daily movement—walk more, take stairs.", "Combi
        }
    }
    jobs = ['Engineer', 'Doctor', 'Teacher', 'Artist', 'Programmer', 'Lawyer', 'Accountant', 'Student', 'Scientist', 'Nurse', 'Driver', 'Chef', 'Farmer'
    areas = ['Urban','Suburban','Rural']
    diets = ['Omnivore','Vegetarian','Vegan','Keto','Mediterranean','Pescatarian']
    ages = list(range(18, 80))
    kb = dict(base_kb)
```

```python
    templates = [
        "For {category} people aged around {age} working as {job} in {area} following {diet} diet: prioritize {focus}. Sample daily change: {sample}.",
        "{category} individuals (job: {job}, area: {area}, diet: {diet}) should consider {focus}. Tip: {sample}.",
        "Personalized plan for {category} > Age {age}, {job}, {area}, {diet}: {focus}. Practical tip: {sample}."
    ]
    focuses = ["increase protein and healthy fats", "reduce refined carbs", "focus on portion control", "add daily 30-min brisk walk", "monitor caloric
    samples = ["add avocado and nuts to snacks", "replace soda with water", "use smaller plates", "walk 8000 steps/day", "track intake with app", "consu
    i = 0
    while True:
        job = random.choice(jobs)
        area = random.choice(areas)
        diet = random.choice(diets)
        age = random.choice(ages)
        category = random.choice(['Underweight','Normal','Overweight'])
        template = random.choice(templates)
        focus = random.choice(focuses)
        sample = random.choice(samples)
        key = f"{category}|job:{job}|area:{area}|diet:{diet}|age:{age}"
        entry = {
            "title": f"{category} guide - {job} - {area} - {diet} - age {age}",
            "advice": [
                template.format(category=category, age=age, job=job, area=area, diet=diet, focus=focus, sample=sample),
                "General: " + " ".join([sample, focus, "Regular check-ups recommended."])
            ]
        }
        kb[key] = entry
```

```python
        kb[key] = entry
        i += 1
        if i % 200 == 0:
            long_text = " ".join([f"Long advice snippet {j}. Keep consistency." for j in range(200)])
            kb[f"BulkText_{i}"] = {"title": "Bulk advice", "advice":[long_text]}
        if i % 500 == 0:
            s = json.dumps(kb, ensure_ascii=False)
            if len(s.encode('utf-8')) > target_bytes: break
        if i > 20000: break

    write_json(kb, r"C:\DATA\kb_healthGuide.json")
    return kb
```

i. Chat-based Health Guide

Xây dựng chatbot dựa trên KB + Prediction Result. Người dùng hỏi → bot trả lời dựa trên BMI dự đoán + hướng dẫn sức khỏe từ KB.

Main

```
# ---------------------------
# Main script orchestration
# ---------------------------
if __name__ == "__main__":
    print("Bắt đầu quy trình phân tích và huấn luyện mô hình...")

    # 1. (a) Tạo dữ liệu và lưu vào C:\DATA\data_4.2.csv
    df = generate_dataset()

    # 2. (b) Hiển thị phân phối dữ liệu
    show_distribution(df)

    # 3. (c) Huấn luyện 5 mô hình ML cơ bản
    ml_results, preprocessor_ml, _, _ = train_basic_ml_models(df)

    # 4. (d) Xây dựng và huấn luyện các mô hình DL
    dl_results = build_and_train_deep_models(df, preprocessor_ml)

    # 5. (e) So sánh và đánh giá tất cả các mô hình
    eval_df, best_name, _, _ = evaluate_all_models(ml_results, dl_results, preprocessor_ml, None, None)

    # 6. (f) Trực quan hóa kết quả và lưu biểu đồ vào C:\DATA\plots/
    visualize_results(df)

    # 7. (h) Xây dựng cơ sở tri thức lớn và lưu vào C:\DATA\kb_healthGuide.json
    kb = create_big_kb()
```

```
# 8. Lưu mô hình tốt nhất và các thành phần tiền xử lý
create_directory('models/')
with open('models/best_model.txt', 'w') as f:
    f.write(best_name)

if best_name in ml_results:
    print(f"Lưu mô hình ML tốt nhất: {best_name}")
    best_pipe = ml_results[best_name]['pipeline']
    save_joblib(best_pipe, f"models/ml_pipeline.joblib")
else:
    print(f"Lưu mô hình DL tốt nhất: {best_name}")
    best_model_dl = dl_results[best_name]['model']
    best_model_dl.save(f"models/dl_model") # TensorFlow format

dl_le = dl_results[next(iter(dl_results))]['label_encoder']
save_joblib(preprocessor_ml, f"models/preprocessor.joblib")
save_joblib(dl_le, f"models/label_encoder.joblib")

print("\nHoàn tất! Các kết quả và mô hình đã được lưu trữ thành công.")
```

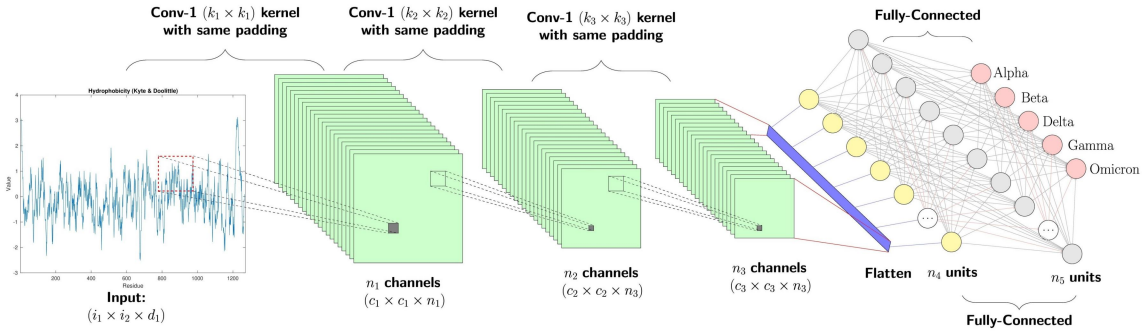
.

1. Cấu trúc CNN, RNN và LSTM

Mạng nơ-ron tích chập (CNN)

CNN là chuyên gia trong việc nhận diện các **mẫu hình trong không gian**, giống như cách chúng ta nhìn thấy các hình ảnh.

> **Cách hoạt động:** Nó giống như một "người kiểm tra" nhỏ (bộ lọc tích chập) lướt qua dữ liệu của bạn, ví dụ như một bức ảnh, để tìm kiếm các đặc điểm cụ thể như đường viền, màu sắc, hoặc hình dạng. Nó tạo ra các bản đồ đặc điểm, sau đó tổng hợp chúng lại để nhận ra các vật thể lớn hơn.



Mạng nơ-ron hồi quy (RNN)

RNN là chuyên gia trong việc ghi nhớ **chuỗi dữ liệu** (dữ liệu có trình tự), giống như cách chúng ta đọc một cuốn sách từ đầu đến cuối để hiểu câu chuyện.

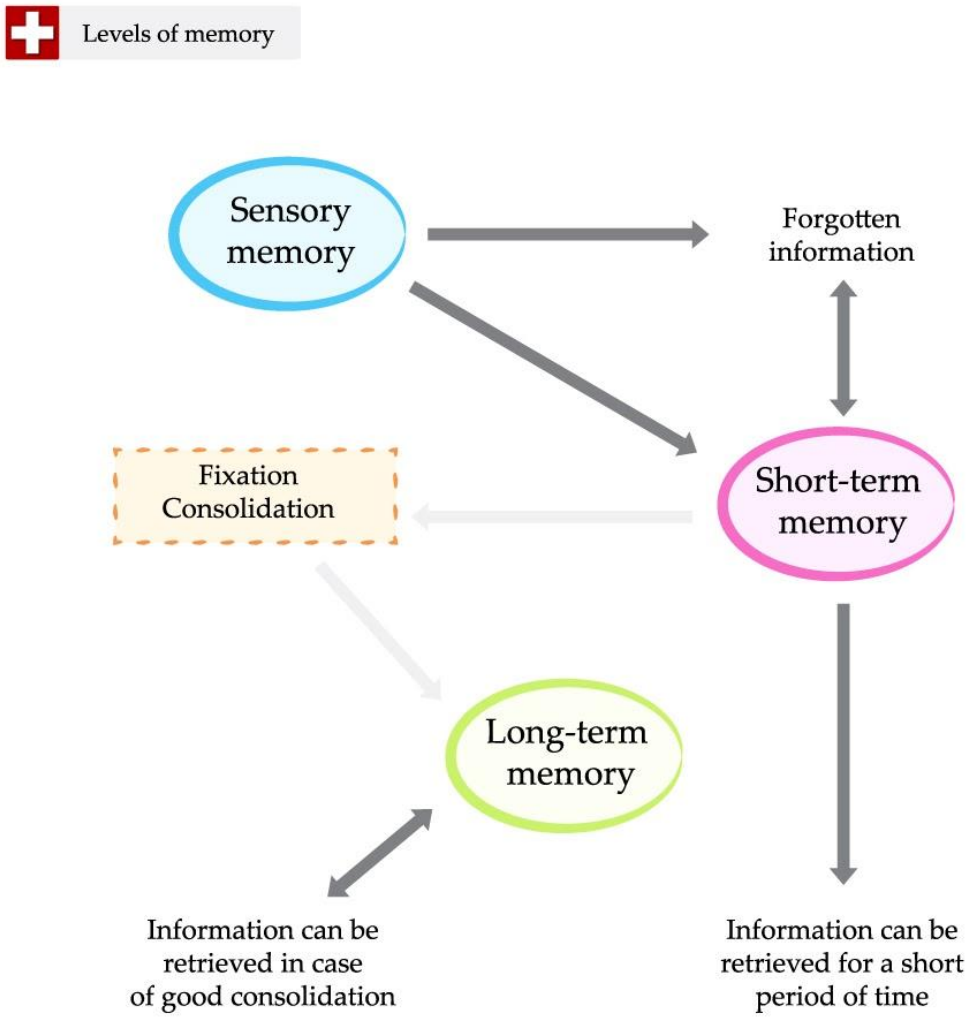
> **Cách hoạt động:** Nó có một "bộ nhớ" để giữ lại thông tin từ các bước trước đó. Mỗi khi nó xử lý một phần dữ liệu mới (ví dụ, một từ trong câu), nó sẽ sử dụng thông tin từ các phần trước đó để đưa ra dự đoán. Tuy nhiên, nó có một điểm yếu là hay "quên" thông tin ở quá xa.

> **Dữ liệu phù hợp:** Văn bản, chuỗi thời gian (giá cổ phiếu, thời tiết), và âm nhạc.

Mạng nơ-ron hồi quy dài-ngắn hạn (LSTM)

LSTM là phiên bản nâng cấp của RNN. Nó giải quyết được điểm yếu về "trí nhớ ngắn hạn" của RNN.

> **Cách hoạt động:** Nó có một "bộ lọc thông minh" bên trong, cho phép nó chủ động quyết định nên ghi nhớ thông tin nào và bỏ qua thông tin nào. Điều này giúp nó xử lý các chuỗi dữ liệu rất dài mà không bị mất thông tin quan trọng



Dữ liệu phù hợp: Tương tự như RNN, nhưng hiệu quả hơn với các chuỗi dài và phức tạp hơn.

Batch, Learning Rate, Epoch và ảnh hưởng của chúng

Batch (Lô dữ liệu)

Batch là một tập hợp nhỏ các mẫu dữ liệu được chọn ngẫu nhiên từ toàn bộ dữ liệu. Mô hình sẽ học trên lô này một lần, sau đó cập nhật các tham số của nó.

Thay đổi Batch:

Batch lớn: Mô hình sẽ cập nhật tham số ít lần hơn nhưng mỗi lần cập nhật chính xác hơn. Quá trình này cần nhiều bộ nhớ hơn.

Batch nhỏ: Mô hình cập nhật tham số thường xuyên hơn, mỗi lần cập nhật có thể không chính xác nhưng giúp mô hình thoát khỏi các "điểm bế tắc" cục bộ (local minima), giúp tổng thể hội tụ tốt hơn.

Learning Rate là "kích thước bước đi" của mô hình khi nó cố gắng tìm ra lời giải tốt nhất.

Thay đổi Learning Rate:

Quá lớn: Mô hình sẽ "bước" quá xa, bỏ qua lời giải tối ưu và có thể không bao giờ tìm thấy nó.

Quá nhỏ: Mô hình sẽ "bước" rất chậm, tốn nhiều thời gian để hội tụ và có thể bị mắc kẹt ở một điểm bế tắc.

Epoch là một chu trình hoàn chỉnh, trong đó mô hình đã xem qua **toàn bộ** dữ liệu huấn luyện một lần.

Thay đổi Epoch:

Quá ít Epoch: Mô hình chưa học đủ, kết quả sẽ kém (underfit).

Quá nhiều Epoch: Mô hình sẽ học thuộc lòng dữ liệu huấn luyện (overfit), dẫn đến kết quả kém khi gặp dữ liệu mới.

2. Biến đổi dữ liệu với `reshape()`

Các mô hình học sâu yêu cầu dữ liệu đầu vào phải có hình dạng (shape) cụ thể. Hàm `reshape()` được sử dụng để định hình lại dữ liệu cho phù hợp.

Đối với CNN (1D): Dữ liệu cần có 3 chiều: `(số_mẫu, chiều_dài, 1)`. Ví dụ, nếu bạn có 10,000 người, mỗi người có 18 đặc trưng, bạn sẽ cần `reshape` nó thành `(10000, 18, 1)`. Chiều thứ ba (`1`) biểu thị số "kênh" dữ liệu, tương tự như kênh màu trong ảnh.

Đối với RNN/LSTM: Dữ liệu cần có 3 chiều: `(số_mẫu, số_bước_thời_gian, số_đặc_trưng)`. Khi mỗi đặc trưng là một "bước thời gian" (ví dụ: tuổi, nghề nghiệp, ...), ta có thể định hình lại tương tự CNN: `(10000, 18, 1)`.

Việc sử dụng `reshape()` đảm bảo rằng dữ liệu của bạn khớp với "kích thước đầu vào" mà mô hình mong đợi.