```
import pandas as pd
from transformers import BertTokenizer, BertForSequenceClassification, Trainer,
TrainingArguments
from datasets import Dataset
import torch
from sklearn.metrics import accuracy_score

# Step 1: Load and View the Data
file_path = '/mnt/data/FOOD-DATA-GROUP1.csv'
df = pd.read_csv(file_path)

# Clean the data (remove unnecessary columns)
df_cleaned = df.drop(columns=['Unnamed: 0', 'Unnamed: 0.1'])

# We will assume that a food item is healthy if its "Nutrition Density" is higher than the median
value
median_nutrition_density = df_cleaned['Nutrition Density'].median()

# Assign a label based on the Nutrition Density: "Healthy" if Nutrition Density > median, else
"Unhealthy"
df_cleaned['label'] = df_cleaned['Nutrition Density'].apply(lambda x: 1 if x >
median_nutrition_density else 0)

# Display the cleaned dataset
print("Cleaned Dataset Preview:")
print(df_cleaned.head())

# Convert the data to a Hugging Face Dataset object
dataset = Dataset.from_pandas(df_cleaned[['food', 'label']])

# Step 2: Tokenizer and Model Setup
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)

# Check for MPS device
device = torch.device("mps" if torch.backends.mps.is_available() else "cpu")
model.to(device)
print(f"Device in use: {device}")

# Tokenize the data (convert food names to tokens that BERT understands)
def tokenize_function(examples):
    return tokenizer(examples['food'], padding="max_length", truncation=True, max_length=128)

# Apply the tokenizer to the dataset
tokenized_dataset = dataset.map(tokenize_function, batched=True)

# Split dataset into train and test sets
train_dataset = tokenized_dataset.train_test_split(test_size=0.2)['train']
test_dataset = tokenized_dataset.train_test_split(test_size=0.2)['test']

# Step 3: Define Metrics (Accuracy)
def compute_metrics(p):
    preds = p.predictions.argmax(axis=-1)  # Get the predicted labels
    labels = p.label_ids  # Get the true labels
    acc = accuracy_score(labels, preds)  # Calculate accuracy
    return {"accuracy": acc}

# Step 4: Train the Model
training_args = TrainingArguments(
    output_dir='./results',          # output directory
    num_train_epochs=3,              # number of training epochs
    per_device_train_batch_size=8,   # batch size for training
    per_device_eval_batch_size=8,    # batch size for evaluation
    evaluation_strategy="epoch",     # evaluate after each epoch
    save_strategy="epoch",           # save model after each epoch
    logging_dir='./logs',            # directory for storing logs
    logging_steps=10,
    load_best_model_at_end=True,     # Save the best model
    metric_for_best_model='accuracy',# Evaluate best model based on accuracy
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,  # Add the metrics function here
)

# Train the model
trainer.train()

# Save the trained model
trainer.save_model('./final_model')

# Evaluate the model
results = trainer.evaluate()
print("Evaluation Results:", results)

# Step 5: Dynamic User Interaction

# Initialize lists to store unique healthy and unhealthy foods
healthy_foods = set()
unhealthy_foods = set()

# Function to make a prediction on a given food item
def predict_health_status(food_item):
    # Tokenize the input
    inputs = tokenizer(food_item, return_tensors="pt", padding=True, truncation=True,
max_length=128).to(device)

    # Make prediction
    with torch.no_grad():
        outputs = model(**inputs)
        prediction = torch.argmax(outputs.logits, dim=-1)

    # Return healthy or unhealthy based on the prediction
    return "Healthy" if prediction.item() == 1 else "Unhealthy"

# Function to continuously ask the user for food items
def user_interaction():
    while True:
        # Prompt user for input
        user_input = input("\nEnter a food item to check its health status (or type 'exit' to quit,
'list' for list of foods): ")

        # Exit condition
        if user_input.lower() == 'exit':
            print("Exiting the program. Goodbye!")
            break

        # Display the list of healthy or unhealthy foods
        elif user_input.lower() == 'list':
            print("\nHealthy Foods List:")
            for food in healthy_foods:
                print(f"- {food}")

            print("\nUnhealthy Foods List:")
            for food in unhealthy_foods:
                print(f"- {food}")

        else:
            # Predict health status
            health_status = predict_health_status(user_input)
            print(f"The health status of '{user_input}' is: {health_status}")

            # Add the food to the appropriate list (ensure no duplicates)
            if health_status == "Healthy":
                healthy_foods.add(user_input)
            else:
                unhealthy_foods.add(user_input)

# Step 6: Start user interaction
user_interaction()
```