```
import pandas as pd

# Load the dataset
data = pd.read_csv("/content/food_coded.csv")

# Display the first few rows of the dataset to inspect column names
print(data.head())

# Data Cleaning - Correct column names if necessary based on the output of data.head()
# Replace 'GPA', 'calories', 'comfort_food_reasons_coded' with actual column names from your dataset
# The error suggests 'calories' is not a column name
# Assuming 'calories_day' is the correct column name based on the dataset
data = data[['GPA', 'calories_day', 'comfort_food_reasons_coded']]  # Keep only relevant columns
data.dropna(subset=['calories_day'], inplace=True)  # Drop rows without abstracts

# Normalize text
def clean_text(text):
    # Check if the input is a string before applying lower()
    if isinstance(text, str):
        text = text.lower()  # Lowercase
        text = ''.join(char for char in text if char.isalnum() or char.isspace())  # Remove special characters
        return text
    else:
        return text  # Return original value if not a string

# Apply clean_text to the 'comfort_food_reasons_coded' column (replace with actual text column name)
data['comfort_food_reasons_coded'] = data['comfort_food_reasons_coded'].apply(clean_text) # Apply to the 'comfort_food_reasons_coded' col
```

```
      GPA  Gender  breakfast  calories_chicken  calories_day  calories_scone  \
0    2.4       2          1               430           NaN           315.0
1  3.654       1          1               610           3.0           420.0
2    3.3       1          1               720           4.0           420.0
3    3.2       1          1               430           3.0           420.0
4    3.5       1          1               720           2.0           420.0

   coffee                     comfort_food        comfort_food_reasons  \
0       1                             none       we dont have comfort
1       2       chocolate, chips, ice cream       Stress, bored, anger
2       2   frozen yogurt, pizza, fast food            stress, sadness
3       2  Pizza, Mac and cheese, ice cream                   Boredom
4       2      Ice cream, chocolate, chips   Stress, boredom, cravings

   comfort_food_reasons_coded  ...  soup  sports  thai_food tortilla_calories  \
0                         9.0  ...   1.0     1.0          1            1165.0
1                         1.0  ...   1.0     1.0          2             725.0
2                         1.0  ...   1.0     2.0          5            1165.0
3                         2.0  ...   1.0     2.0          5             725.0
4                         1.0  ...   1.0     1.0          4             940.0

   turkey_calories  type_sports veggies_day  vitamins  waffle_calories  \
0              345   car racing           5         1             1315
1              690  Basketball            4         2              900
2              500         none           5         1              900
3              690          NaN           3         1             1315
4              500     Softball           4         2              760

                      weight
0                        187
1                        155
2  I'm not answering this.
3             Not sure, 240
4                        190

[5 rows x 61 columns]
```

```
from sklearn.model_selection import train_test_split

# Split the dataset into train and test sets
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)  # 80% train, 20% test



# Apply clean_text to the 'comfort_food_reasons_coded' column (replace with actual text column name)
data['comfort_food_reasons_coded'] = data['comfort_food_reasons_coded'].apply(clean_text) # Apply to the 'comfort_food_reasons_coded' co

# Ensure all values in the column are strings before tokenization
data['comfort_food_reasons_coded'] = data['comfort_food_reasons_coded'].astype(str)

# Split the dataset into train and test sets
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)  # 80% train, 20% test


# Load the tokenizer
tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')
```

```
# Tokenize the input texts
train_encodings = tokenizer(train_data['comfort_food_reasons_coded'].tolist(), truncation=True, padding=True)
test_encodings = tokenizer(test_data['comfort_food_reasons_coded'].tolist(), truncation=True, padding=True)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenization_spaces`
  warnings.warn(
```

```
import torch
from torch.utils.data import Dataset
import pandas as pd

class ComfortFoodDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        # Convert labels to float before creating tensor
        item['labels'] = torch.tensor(float(self.labels[idx]))
        return item

    def __len__(self):
        return len(self.encodings['input_ids'])

# Create datasets
# Ensure GPA values are converted to float, handling non-numeric values
# Replace non-numeric values with NaN
train_data['GPA'] = pd.to_numeric(train_data['GPA'], errors='coerce')
test_data['GPA'] = pd.to_numeric(test_data['GPA'], errors='coerce')

# Drop rows with NaN in 'GPA' column if needed
train_data.dropna(subset=['GPA'], inplace=True)
test_data.dropna(subset=['GPA'], inplace=True)

train_labels = train_data['GPA'].astype(float).tolist()
test_labels = test_data['GPA'].astype(float).tolist()
train_dataset = ComfortFoodDataset(train_encodings, train_labels)
test_dataset = ComfortFoodDataset(test_encodings, test_labels)
```

```
import torch
from torch.utils.data import Dataset
import pandas as pd
from transformers import AutoTokenizer

class ComfortFoodDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        # Convert labels to float before creating tensor
        item['labels'] = torch.tensor(float(self.labels[idx]))
        return item

    def __len__(self):
        return len(self.encodings['input_ids'])

# Create datasets
# Ensure GPA values are converted to float, handling non-numeric values
# Replace non-numeric values with NaN
train_data['GPA'] = pd.to_numeric(train_data['GPA'], errors='coerce')
test_data['GPA'] = pd.to_numeric(test_data['GPA'], errors='coerce')

# Drop rows with NaN in 'GPA' column if needed
train_data.dropna(subset=['GPA'], inplace=True)
test_data.dropna(subset=['GPA'], inplace=True)

# Recreate train_encodings and test_encodings after dropping rows
tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')  # Load tokenizer
train_encodings = tokenizer(train_data['comfort_food_reasons_coded'].tolist(), truncation=True, padding=True)
test_encodings = tokenizer(test_data['comfort_food_reasons_coded'].tolist(), truncation=True, padding=True)

train_labels = train_data['GPA'].astype(float).tolist()
test_labels = test_data['GPA'].astype(float).tolist()
train_dataset = ComfortFoodDataset(train_encodings, train_labels)
test_dataset = ComfortFoodDataset(test_encodings, test_labels)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenization_spaces`
  warnings.warn(
```

```
import pandas as pd
import torch
from torch.utils.data import Dataset
from transformers import AutoTokenizer

# Load the dataset
data = pd.read_csv("/content/food_coded.csv")

# Data Cleaning - Keep only relevant columns and drop NaN values
data = data[['GPA', 'calories_day', 'comfort_food_reasons_coded']]
data.dropna(subset=['calories_day', 'GPA', 'comfort_food_reasons_coded'], inplace=True)

# Normalize text
def clean_text(text):
    if isinstance(text, str):
        text = text.lower()  # Lowercase
        text = ''.join(char for char in text if char.isalnum() or char.isspace())  # Remove special characters
        return text
    return ""  # Return empty string if not a string

# Apply clean_text to the 'comfort_food_reasons_coded' column
data['comfort_food_reasons_coded'] = data['comfort_food_reasons_coded'].apply(clean_text)

# Create a custom dataset class for Hugging Face Trainer
class ComfortFoodDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(float(self.labels[idx]))  # Convert labels to float
        return item

    def __len__(self):
        return len(self.encodings['input_ids'])

# Convert GPA values to float, handling non-numeric values
data['GPA'] = pd.to_numeric(data['GPA'], errors='coerce')
data.dropna(subset=['GPA'], inplace=True)

# Split the dataset into training and testing sets (e.g., 80% train, 20% test)
train_data = data.sample(frac=0.8, random_state=42)  # Randomly sample 80% of the data
test_data = data.drop(train_data.index)  # The remaining 20%

# Load tokenizer
tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')

# Ensure all entries are strings and tokenize
train_encodings = tokenizer(train_data['comfort_food_reasons_coded'].astype(str).tolist(),
                            truncation=True,
                            padding=True)
test_encodings = tokenizer(test_data['comfort_food_reasons_coded'].astype(str).tolist(),
                           truncation=True,
                           padding=True)

# Create datasets
train_labels = train_data['GPA'].astype(float).tolist()
test_labels = test_data['GPA'].astype(float).tolist()
train_dataset = ComfortFoodDataset(train_encodings, train_labels)
test_dataset = ComfortFoodDataset(test_encodings, test_labels)

# Display the first few entries of the training dataset for verification
print("Sample of training dataset:")
print(train_dataset[0])  # Print the first item in the training dataset
```

```
Sample of training dataset:
{'input_ids': tensor([101, 102]), 'token_type_ids': tensor([0, 0]), 'attention_mask': tensor([1, 1]), 'labels': tensor(3.2000)}
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenization_spaces`
  warnings.warn(
```

```
from transformers import AutoModelForSequenceClassification, Trainer, TrainingArguments

# Load the model
model = AutoModelForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=1)  # Use num_labels=1 for regression tasks
```

```
# Define training arguments
training_args = TrainingArguments(
    output_dir='./results',              # output directory
    num_train_epochs=3,                  # total number of training epochs
    per_device_train_batch_size=8,       # batch size per device during training
    per_device_eval_batch_size=16,       # batch size for evaluation
    warmup_steps=500,                     # number of warmup steps for learning rate scheduler
    weight_decay=0.01,                   # strength of weight decay
    logging_dir='./logs',                 # directory for storing logs
    logging_steps=10,
)

# Initialize Trainer
trainer = Trainer(
    model=model,                         # the instantiated 🤗 Transformers model to be trained
    args=training_args,                  # training arguments, defined above
    train_dataset=train_dataset,         # training dataset
    eval_dataset=test_dataset             # evaluation dataset
)

# Train the model
trainer.train()

# Evaluate the model
trainer.evaluate()
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly init
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```


Step	Training Loss
10	11.358600
20	10.284900


```
{'eval_loss': 8.159689903259277,
 'eval_runtime': 0.3841,
 'eval_samples_per_second': 44.254,
 'eval_steps_per_second': 5.206,
 'epoch': 3.0}
```