In [1]:

```python
import pandas as pd
import numpy as np

import torch
from torch import nn
from torch.utils.data import Dataset, DataLoader, RandomSampler, SequentialSampler
from transformers import BertForTokenClassification, BertTokenizer
import datasets
from argparse import Namespace, ArgumentParser
import json
import os

from sklearn.metrics import accuracy_score
from tqdm import tqdm
from datetime import datetime
import csv


```

```
/Users/zl/miniconda3/envs/ebay/lib/python3.10/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not fou
nd. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
(https://ipywidgets.readthedocs.io/en/stable/user_install.html)
  from .autonotebook import tqdm as notebook_tqdm
```

In [2]:

```python
args = Namespace(
    train_data_path='./2023 eBay ML Challenge Data/Train_Tagged_Titles.tsv',
    test_data_path='./2023 eBay ML Challenge Data/Listing_Titles.tsv',

    device='mps',

    # train test split ratio
    split_ratio=(0.75, 0.15, 0.1),
    # tokenizer vector max length
    max_length=70,
    # dataloader num of workers
    num_workers=0,
    # batch size
    batch_size=128,
    # num of classes
    n_classes=37,
    # drop out rate in NERModel
    # drop_rate=0.5,

    # early stopping threshold
    early_stopping_threshold=10,

    max_epochs=1000,

    lr=2e-5,
    # used when clipping gradient
    max_grad_norm=10,
    seed=1314

)
```

In [3]:

```python
with open('commandline_args.txt', 'w') as f:
    json.dump(args.__dict__, f, indent=2)


with open('commandline_args.txt', 'r') as f:
    args2 = Namespace(**json.load(f))

print(args2)
```

```
Namespace(train_data_path='./2023 eBay ML Challenge Data/Train_Tagged_Titles.tsv', test_data_path='./2023 e
Bay ML Challenge Data/Listing_Titles.tsv', device='mps', split_ratio=[0.75, 0.15, 0.1], max_length=70, num_
workers=0, batch_size=128, n_classes=37, early_stopping_threshold=10, max_epochs=1000, lr=2e-05, max_grad_n
orm=10, seed=1314)
```

In [ ]:

```python

```

In [4]:

```python
def load_train_data(args):
    tagged = pd.read_csv(args.train_data_path, sep='\t', dtype=str, quoting=csv.QUOTE_NONE, na_values='',

    # titles = pd.read_csv('./2023 eBay ML Challenge Data/Listing_Titles.tsv', sep='\t', quoting=3)
    tagged.fillna('nan', inplace=True)
    # tagged.loc[((tagged['Tag'] == 'No Tag') | (tagged['Tag'] == 'Obscure')), 'Tag'] = 'No Tag'
    # create tag hashtable
    id2tag = {0:'No Tag'}
    tag2id = {'No Tag':0}
    i = 1
    for tag in np.sort(tagged['Tag'].unique()):

        if tag != 'No Tag':
            id2tag[i] = tag
            tag2id[tag] = i
            i+=1
    # add tag idx column
    tagged['label'] = tagged['Tag'].apply(lambda x: tag2id[x])
    return tagged, id2tag, tag2id

def load_test_data(args):
    titles = pd.read_csv(args.test_data_path, sep='\t', quoting=3, na_values='', keep_default_na=False)

    return titles


def train_test_split(args, tagged):
    unique_record = tagged['Record Number'].unique()
    np.random.seed(args.seed)
    train_val_test_idx = np.random.choice(unique_record, size=len(unique_record), replace=False)

    record_count = len(train_val_test_idx)
    ratio = args.split_ratio
    bound1 = round(ratio[0] * record_count)
    bound2 = round((ratio[0] + ratio[1]) * record_count)
    train_df = tagged[tagged['Record Number'].isin(train_val_test_idx[: bound1])]

    val_df = tagged[tagged['Record Number'].isin(train_val_test_idx[bound1:bound2])]
    test_df = tagged[tagged['Record Number'].isin(train_val_test_idx[bound2:])]

    print(f"Splited train: {len(train_df)}, val: {len(val_df)}, test: {len(test_df)}")
    return train_df, val_df, test_df

```

In [5]:

```python
tagged, id2tag, tag2id = load_train_data(args)
tagged
```

Out[5]:

	Record Number	Title	Token	Tag	label
0	1	Supreme Nike SB Dunk High By any Means Red US1...	Supreme	Modell	21
1	1	Supreme Nike SB Dunk High By any Means Red US1...	Nike	Marke	19
2	1	Supreme Nike SB Dunk High By any Means Red US1...	SB	Produktlinie	26
3	1	Supreme Nike SB Dunk High By any Means Red US1...	Dunk	nan	36
4	1	Supreme Nike SB Dunk High By any Means Red US1...	High	Schuhschaft-Typ	27
...	...	...	...	...	...
55178	5000	Herren Trekking Schuhe Outdoor Sneaker Sportsc...	Sportschuhe	Produktart	25
55179	5000	Herren Trekking Schuhe Outdoor Sneaker Sportsc...	Wanderschuh	nan	36
55180	5000	Herren Trekking Schuhe Outdoor Sneaker Sportsc...	Big	No Tag	0
55181	5000	Herren Trekking Schuhe Outdoor Sneaker Sportsc...	Size	No Tag	0
55182	5000	Herren Trekking Schuhe Outdoor Sneaker Sportsc...	U37	No Tag	0

55183 rows × 5 columns

In [6]:

```
id2tag
```

Out[6]:

```
{0: 'No Tag',
 1: 'Abteilung',
 2: 'Aktivität',
 3: 'Akzente',
 4: 'Anlass',
 5: 'Besonderheiten',
 6: 'Charakter',
 7: 'Charakter Familie',
 8: 'Dämpfungsgrad',
 9: 'EU-Schuhgröße',
 10: 'Erscheinungsjahr',
 11: 'Farbe',
 12: 'Futtermaterial',
 13: 'Gewebeart',
 14: 'Herstellernummer',
 15: 'Herstellungsland und -region',
 16: 'Innensohlenmaterial',
 17: 'Jahreszeit',
 18: 'Laufsohlenmaterial',
 19: 'Marke',
 20: 'Maßeinheit',
 21: 'Modell',
 22: 'Muster',
 23: 'Obermaterial',
 24: 'Obscure',
 25: 'Produktart',
 26: 'Produktlinie',
 27: 'Schuhschaft-Typ',
 28: 'Schuhweite',
 29: 'Stil',
 30: 'Stollentyp',
 31: 'Thema',
 32: 'UK-Schuhgröße',
 33: 'US-Schuhgröße',
 34: 'Verschluss',
 35: 'Zwischensohlen-Typ',
 36: 'nan'}
```

In [7]:

```
# -----embed_data
# {'input_ids': [101, 5256, 2033, 2039, 2012, 3157, 2572, 2006, 5958, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
# ----example
# {'id': '1', 'label': 48, 'label_text': 'alarm_set', 'text': 'wake me up at nine am on friday'}

def tokenize_and_preserve_labels(sentence, label, tokenizer):
    # label: list of label
    # labels: list of label match the tokenized word
    # lab: element in label

    tokenized_sentence = []
    labels = []

    sentence = sentence.strip()

    for word, lab in zip(sentence.split(), label):

        # Tokenize the word and count # of subwords the word is broken into
        tokenized_word = tokenizer.tokenize(word)
        n_subwords = len(tokenized_word)

        # Add the tokenized word to the final tokenized word list
        tokenized_sentence.extend(tokenized_word)

        # Add the same label to the new list of labels `n_subwords` times
        labels.extend([lab] * n_subwords)

    return tokenized_sentence, labels

class TitlesDataset(Dataset):
    def __init__(self, args: Namespace, data: pd.DataFrame, tokenizer, split='train'):
        def compress(x):
            return list(x)


        self.tokenizer = tokenizer
        self.data = data.groupby(['Record Number', 'Title']).agg(compress).reset_index()
        self.max_len = args.max_length
        self.split = split

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        # step 1: tokenize (and adapt corresponding labels)
        sentence = self.data['Title'][index]

        label = self.data['label'][index]
        tokenized_sentence, labels = tokenize_and_preserve_labels(sentence, label, self.tokenizer)

        # step 2: add special tokens (and corresponding labels)

        tokenized_sentence = ["[CLS]"] + tokenized_sentence + ["[SEP]"] # add special tokens
        labels.insert(0, tag2id['No Tag']) # add outside label for [CLS] token
        labels.insert(-1, tag2id['No Tag']) # add outside label for [SEP] token

        # step 3: truncating/padding
        maxlen = self.max_len

        if (len(tokenized_sentence) > maxlen):
            # truncate (right)
            tokenized_sentence = tokenized_sentence[:maxlen]
            labels = labels[:maxlen]
        else:
            # pad
            tokenized_sentence = tokenized_sentence + ['[PAD]'for _ in range(maxlen - len(tokenized_sent
            labels = labels + [tag2id['No Tag'] for _ in range(maxlen - len(labels))]

        # step 4: obtain the attention mask
        attn_mask = [1 if tok != '[PAD]' else 0 for tok in tokenized_sentence]

        # step 5: convert tokens to input ids
        ids = self.tokenizer.convert_tokens_to_ids(tokenized_sentence)

        # label_ids = [label2id[label] for label in labels]
        # the following line is deprecated
        #label_ids = [label if label != 0 else -100 for label in label_ids]
        return {
              'input_ids': torch.tensor(ids, dtype=torch.long),
              'attention_mask': torch.tensor(attn_mask, dtype=torch.long),
              #'token_type_ids': torch.tensor(token_ids, dtype=torch.long),
              'labels': torch.tensor(labels, dtype=torch.long)
        }

#     def collate_func(self, batch):
#         input_ids = torch.tensor([f['input_ids'] for f in batch], dtype=torch.long)
#         token_type_ids = torch.tensor([f['token_type_ids'] for f in batch], dtype=torch.long)
#         attention_mask = torch.tensor([f['attention_mask'] for f in batch], dtype=torch.long)
#         label = torch.tensor([f['label'] for f in batch], dtype=torch.long)

#         return input_ids, token_type_ids, attention_mask, label

def get_dataloader(args, dataset):
    # collate = dataset.collate_func
    # sampler = RandomSampler(dataset) if dataset.split == 'train' else SequentialSampler(dataset)

    # dataloader = DataLoader(dataset, sampler=sampler, batch_size=args.batch_size, num_workers=args.num
    if dataset.split == 'test':
        shuffle = False
    else:
        shuffle = True

    dataloader = DataLoader(dataset, batch_size=args.batch_size, shuffle=shuffle, num_workers=args.num_w

    print(f"Loaded {dataset.split} data with {len(dataloader)} batches, each batch {args.batch_size} ins
    return dataloader




```

In [8]:

```python
def train(args, model, train_dataloader, val_dataloader, optimizer):
    dt_string = datetime.now().strftime("%Y-%m-%d-%H")
    path = f'./models/{dt_string}'
    if not os.path.exists(path):
        os.makedirs(path)
    with open(os.path.join(path,'args.txt') , 'w') as f:
        json.dump(args.__dict__, f, indent=2)

    model_path = os.path.join(path,'best_model.pt')

    output_stats = {'val_losses': [], 'val_f1': []}
    min_loss = float('inf')
    threshold = args.early_stopping_threshold
    for epoch_count in range(args.max_epochs):

        tr_loss, tr_accuracy = 0, 0
        # put model in training mode
        model.train()
        with tqdm(total=len(train_dataloader)) as p:
            for idx, batch in enumerate(train_dataloader):
                input_ids = batch['input_ids'].to(args.device, dtype = torch.long)
                attention_mask = batch['attention_mask'].to(args.device, dtype = torch.long)
                labels = batch['labels'].to(args.device, dtype = torch.long)

                outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)

                loss, tr_logits = outputs.loss, outputs.logits
                tr_loss += loss.item()

                # compute training accuracy
                flattened_targets = labels.view(-1) # shape (batch_size * seq_len,)
                active_logits = tr_logits.view(-1, model.num_labels) # shape (batch_size * seq_len, num_
                flattened_predictions = torch.argmax(active_logits, axis=1) # shape (batch_size * seq_le
                # now, use mask to determine where we should compare predictions with targets (includes
                active_accuracy = attention_mask.view(-1) == 1 # active accuracy is also of shape (batch


                labels = torch.masked_select(flattened_targets, active_accuracy)
                predictions = torch.masked_select(flattened_predictions, active_accuracy)

                batch_tr_accuracy = accuracy_score(labels.cpu().numpy(), predictions.cpu().numpy())
                tr_accuracy += batch_tr_accuracy

                # backward pass
                optimizer.zero_grad()
                loss.backward()
                # gradient clipping
                torch.nn.utils.clip_grad_norm_(
                    parameters=model.parameters(), max_norm=args.max_grad_norm
                )
                optimizer.step()

                p.set_postfix({'bcls': round(loss.item(), 3), 'bcacc': round(batch_tr_accuracy, 3)})
                p.update()

        epoch_loss = tr_loss / len(train_dataloader)
        tr_accuracy = tr_accuracy / len(train_dataloader)
        print(f"Training loss epoch {epoch_count}: {epoch_loss}")
        print(f"Training accuracy epoch: {tr_accuracy}")

        val_loss, val_f1 = valid(args, val_dataloader, model, return_pred=False)
        output_stats['val_losses'].append(val_loss)
        output_stats['val_f1'].append(val_f1)

        if val_loss < min_loss:
            min_loss = val_loss
            threshold = args.early_stopping_threshold
            torch.save(model.state_dict(), model_path)
        else:
            threshold -= 1
            if threshold == 0:
                return output_stats

    return output_stats

def valid(args, val_dataloader, model, return_pred=False):
    # put model in evaluation mode
    model.eval()

    eval_loss, eval_accuracy = 0, 0
    final_tokens, final_predictions = [], []

    eval_preds, eval_labels = [], []
    with torch.no_grad():
        pbar = tqdm(enumerate(val_dataloader), total=len(val_dataloader))
        for idx, batch in pbar:

            input_ids = batch['input_ids'].to(args.device)

            attention_mask = batch['attention_mask'].to(args.device)
            labels = batch['labels'].to(args.device)

            outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
            loss, eval_logits = outputs.loss, outputs.logits

            eval_loss += loss.item()

            # compute evaluation accuracy
            flattened_targets = labels.view(-1) # shape (batch_size * seq_len,)
            active_logits = eval_logits.view(-1, model.num_labels) # shape (batch_size * seq_len, num_la
            flattened_predictions = torch.argmax(active_logits, axis=1) # shape (batch_size * seq_len,)

            # now, use mask to determine where we should compare predictions with targets (includes [CLS
            active_accuracy = attention_mask.view(-1) == 1 # active accuracy is also of shape (batch_siz
            labels = torch.masked_select(flattened_targets, active_accuracy)
            predictions = torch.masked_select(flattened_predictions, active_accuracy)

            eval_labels.extend(labels)
            eval_preds.extend(predictions)

            batch_eval_accuracy = accuracy_score(labels.cpu().numpy(), predictions.cpu().numpy())
            eval_accuracy += batch_eval_accuracy


            if return_pred:

                # final_labels.append([id2tag[idd.item()] for idd in labels])
                input_id_each = []
                for i in input_ids:

                    input_id_each.extend(tokenizer.convert_ids_to_tokens(i.tolist()))
                final_tokens.append(input_id_each)

                final_predictions.append([id2tag[idd.item()] for idd in flattened_predictions.cpu().nump

            pbar.set_postfix({'eval bcls': loss.item(), 'eval bcacc': batch_eval_accuracy})
            pbar.update()


    eval_loss = eval_loss / len(val_dataloader)
    eval_accuracy = eval_accuracy / len(val_dataloader)
    eval_f1 = weighted_F1(args, weight, torch.tensor(eval_labels).cpu().numpy(), torch.tensor(eval_preds
    print(f"Validation Loss: {eval_loss}")
    print(f"Validation Accuracy: {eval_accuracy}")
    print(f"Validation F1: {eval_f1}")
    if return_pred:
        return final_tokens, final_predictions, eval_loss, eval_f1
    else:
        return eval_loss, eval_f1


def weighted_F1(args, weight, labels, predictions):
    # f1_score(y_true, y_pred, average='weighted')
    # want to use all weights from all data
    final_f1 = 0
    for i in range(1, args.n_classes):
        lab_sub = (labels == i)
        pred_sub = (predictions == i)
        inter = (lab_sub & pred_sub).sum()
        if inter == 0:
            f1 = 0
        else:
            precision = inter / pred_sub.sum()
            recall = inter / lab_sub.sum()
            if (precision + recall) == 0:
                f1 = 0
            else:
                f1 = 2 * precision * recall / (precision + recall)

        weighted_f1 = weight[i] * f1
        final_f1 += weighted_f1
    return final_f1

```

In [9]:

```
torch.tensor([[1,2,3],[2.0,3,4]]).squeeze()
```

Out[9]:

```
tensor([[1., 2., 3.],
        [2., 3., 4.]])
```

```
from transformers import AutoTokenizer
from transformers import AutoModelForTokenClassification

```

In [9]:

```
# main
tagged, id2tag, tag2id = load_train_data(args)
weight = tagged['label'][tagged['label']!=0].value_counts(normalize=True)
train_df, val_df, test_df = train_test_split(args, tagged)
tokenizer = BertTokenizer.from_pretrained("bert-base-german-cased", truncation_side="right")
# tokenizer = AutoTokenizer.from_pretrained("microsoft/deberta-v3-large", use_fast=False)
model = BertForTokenClassification.from_pretrained("bert-base-german-cased",num_labels=args.n_classes,
                                                   id2label=id2tag,
                                                   label2id=tag2id)
# model = BertForTokenClassification.from_pretrained("microsoft/deberta-v3-large",num_labels=args.n_class
#                                                    id2label=id2tag,
#                                                    label2id=tag2id)
model.to(args.device)

train_dataset = TitlesDataset(args, train_df, tokenizer, split='train')
val_dataset = TitlesDataset(args, val_df, tokenizer, split='val')
test_dataset = TitlesDataset(args, test_df, tokenizer, split='test')

train_dataloader = get_dataloader(args, train_dataset)
val_dataloader = get_dataloader(args, val_dataset)
test_dataloader = get_dataloader(args, test_dataset)

optimizer = torch.optim.AdamW(params=model.parameters(), lr=args.lr)
```

```
Splited train: 41377, val: 8259, test: 5547

Some weights of the model checkpoint at bert-base-german-cased were not used when initializing BertForToken
Classification: ['cls.predictions.transform.LayerNorm.weight', 'cls.predictions.decoder.weight', 'cls.predi
ctions.bias', 'cls.predictions.transform.dense.bias', 'cls.seq_relationship.weight', 'cls.seq_relationship.
bias', 'cls.predictions.transform.LayerNorm.bias', 'cls.predictions.transform.dense.weight']
- This IS expected if you are initializing BertForTokenClassification from the checkpoint of a model traine
d on another task or with another architecture (e.g. initializing a BertForSequenceClassification model fro
m a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForTokenClassification from the checkpoint of a model th
at you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSeq
uenceClassification model).
Some weights of BertForTokenClassification were not initialized from the model checkpoint at bert-base-germ
an-cased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inferen
ce.

Loaded train data with 30 batches, each batch 128 instances
Loaded val data with 6 batches, each batch 128 instances
Loaded test data with 4 batches, each batch 128 instances
```

In [ ]:

```
train_output = train(args, model, train_dataloader, val_dataloader, optimizer )
```

In [11]:

```
args.n_classes
```

Out[11]:

```
37
```

In [12]:

```
model_inference = BertForTokenClassification.from_pretrained("bert-base-german-cased",num_labels= args.n_
                                                   id2label=id2tag,
                                                   label2id=tag2id)
model_inference.load_state_dict(torch.load('models/2023-11-11-00/best_model.pt'))
# model_inference.load_state_dict(torch.load('models/best_model_2023-05-19-05.pt'))
model_inference.to(args.device)
```

```
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=37, bias=True)
)
```

In [14]:

```
# sentence = 'Supreme Nike SB Dunk High By any Means Red US1'
def inference(sentence):
    inputs = tokenizer(sentence, padding='max_length', truncation=True, max_length=args.max_length, retur

    # move to gpu
    ids = inputs["input_ids"].to(args.device)
    mask = inputs["attention_mask"].to(args.device)
    # forward pass
    outputs = model_inference(ids, mask)
    logits = outputs[0]

    active_logits = logits.view(-1, model_inference.num_labels) # shape (batch_size * seq_len, num_labels
    flattened_predictions = torch.argmax(active_logits, axis=1) # shape (batch_size*seq_len,) - predictic

    tokens = tokenizer.convert_ids_to_tokens(ids.squeeze().tolist())

    token_predictions = [id2tag[i] for i in flattened_predictions.cpu().numpy()]
    wp_preds = list(zip(tokens, token_predictions)) # list of tuples. Each tuple = (wordpiece, predictior

    word_level_predictions = []
    for pair in wp_preds:
        if (pair[0].startswith("##")) or (pair[0] in ['[CLS]', '[SEP]', '[PAD]']):
        # skip prediction
            continue
        else:
            word_level_predictions.append(pair[1])

    # we join tokens, if they are not special ones
    str_rep = " ".join([t[0] for t in wp_preds if t[0] not in ['[CLS]', '[SEP]', '[PAD]']]).replace(" ##'
    return word_level_predictions, str_rep


```

In [ ]:

```

```

In [16]:

```
test_results = valid(args, test_dataloader, model_inference, return_pred=True)
```

```
100%|██████████| 4/4 [00:02<00:00,  1.80it/s, eval bcls=0.169, eval bcacc=0.886]

Validation Loss: 0.18057096749544144
Validation Accuracy: 0.880394787740071
Validation F1: 0.8642013242690667
```

In [ ]:

```
sub['prediction'] = sub['Title'].apply(inference)
sub
```

In [17]:

```
for i in range(len(test_results[0])):
    tokens = test_results[0][i]
    token_predictions = test_results[1][i]
    wp_preds = list(zip(tokens, token_predictions)) # list of tuples. Each tuple = (wordpiece, predictior
    word_level_predictions = []
    word_level_input = []
    for j in range(len(wp_preds)):
        pair = wp_preds[j]
        if pair[0] == '[CLS]':
            each_item = []
            each_input = []
            for k in range(j+1, len(wp_preds)):

                inside_pair = wp_preds[k]

                if inside_pair[0] == '[SEP]':
                    word_level_predictions.append(each_item)

                    word_level_input.append(' '.join(each_input).replace(' ##', ''))
                    break
                else:
                    each_input.append(inside_pair[0])

                    if (inside_pair[0].startswith("##")):
                        continue
                    else:
                        each_item.append(inside_pair[1])



```

In [ ]:

```

```

In [18]:

```
pd.DataFrame({'Description': word_level_input, 'NER': word_level_predictions})
```

Out[18]:

	Description	NER
0	Balensiaga Triple S Gr . 40	[Marke, Modell, nan, No Tag, EU-Schuhgröße, No...
1	adidas ZX Flux XENO Frozen Yellow (2015) EU ...	[Marke, Modell, nan, nan, nan, nan, No Tag, Er...
2	Sneaker Textilschuhe Hochschaft Stoffschuhe Ca...	[Stil, Produktart, No Tag, Produktart, Gewebea...
3	Adidas Runfalcon 2 . 0 Herren Schuhe Sneaker T...	[Marke, Modell, nan, nan, nan, Abteilung, Prod...
4	Adidas Originals OZWEEGO Sneakers , Gr . 42 2 ...	[Marke, Produktlinie, Modell, Stil, No Tag, No...
...	...	...
111	NEU Converse JP Mid Gr . 44 , 5 Chucks Schuhe ...	[No Tag, Marke, Modell, Schuhschaft-Typ, No Ta...
112	oft getragene damenschuhe abgenutzt	[No Tag, No Tag, Produktart, No Tag]
113	Damen Sneaker Freizeitschuhe Footflexx Komfort...	[Abteilung, Stil, Produktart, Marke, Produktar...
114	New Balance M 770 KGR 40 40 , 5 45 , 5 made in...	[Marke, nan, Modell, Modell, nan, EU-Schuhgröß...
115	SONRA Proto DHL 90 / 300 Neu 44	[Marke, Modell, nan, Modell, No Tag, nan, No T...

116 rows × 2 columns

In [ ]:

```

```