I. Crawling

I have chosen task 2 which consists of creating a ner model to get furniture product names from a list of urls. Because of the length of the document and the need to parse the urls fast I have decided to go with scrapy module because it supports concurrent processing of multiple requests and responses.

The csv with urls is saved under the name **pages.csv.**

I created a new scrapy project called **furniturescraper**, using the command : “scrapy startproject furniturescraper” and then created the spider, called **furniturespider**, using the command “scrapy genspider furniturespider”.

In the file furniturespider.py I have the method **start_requests()** which initiates the web scraping process by sending requests to the specified URLs. At first I started by scraping the first 100 urls but because many urls had no meaningful response (many 404 urls) and the output file had about 50 urls out of 100 scraped , I have done the scrape with first 200 urls.

I wanted to double check the url response so I created the file **check_reposnse.py** which just printed each url and its status code and in the end prints the number of urls with status code=200 out of the first 100 urls. By this looks like the spider performed well.

Going back to furniturespider , the next method in the file **parse()**: Parses the response from each URL and extracts the content. I first tried by getting all data from the page but a lot was redundant and my output was getting too large with too few product names compared to the full retrieved text. So I retrieved only text from header 1, header 2, header 3, spans and paragraphs (p), and joined them all after.

For text cleaning, I removed extra spaces, I lowered all text, I kept only words longer than one letter and shorter than 14 because sometimes some words got concatenated together and were not meaningful. I also removed stopwords and removed numeric data. I have also seen that some texts were in foreign languages so I used the Translator from googletrans library and translated to english the texts.

The output was saved in a json file with “url” and “text_content” corresponding to each.

I have also tried using lemmatization because I wanted to remove plurals and see if my NER model would work better, but it did not so I commented it, as it can be seen from the code. The results comparison is at the end of the documentation.

To get the output I used the command “scrapy crawl furniturespider -O full_path/data/output.json”. In the data folder are stored all data outputs.

II. Annotation

For the annotation part I created a new file called **product_names.py** where I have different furniture names gotten from both searching on internet and from chatGpt. This files consists of 3 sets: one_word_furniture, two_word_furniture, three_word_furniture.

In the notebook **train_data_annotation.ipynb** I use first the function called **extract_products(text)** which gives the output in **'data/produse_in_urls.json'.** The point of this function is to get in a new json file the products present in the text_content for each url:


```
    {
        "url": "https://www.hudsonfurniture.com.au/products/string-weave-timber-stool",
        "text_content": "string weave timber stool related products mcha new string weave stool made sungkai wood super comfy lightweight perfect
fit kitchen bench dimensions need know width cm depth cm height cm sign save subscribe get special offers free giveaways onceinalifetime deals
hudson furniture close menu instagram facebook twitter pinterest close cart site navigation log cart search close esc string weave timber stool
regular price add cart share share facebook tweet tweet twitter pin pin pinterest quick view quick view quick view quick view quick view quick
view close esc close esc close esc close esc close esc close esc subscribe instagram facebook twitter pinterest previous next close esc",
        "products": [
            "stool",
            "stool",
            "bench",
            "stool"
        ]
    },
```


The logic behind this function is that it parses through each word from the text_content, then searches first if it and the next 2 words are in a 3-word furniture name, then if it and the next word is in a 2-word furniture name, and finally if it's in the single word furniture name set. If a match for 3-word or 2-word is found, it skips the rest of the words from the 3-word or 2-word group. It returns then a list of the discovered products.

The next function is called **annotate_products(products).** This function annotates the products with their according labels. For each product in products it splits them into words. If there is a single word then the word gets the annotation B-Product . If there are multiple words, the first one gets B-Product and the other get I-Product. The data is saved then in a new json file called **'data/produse_in_urls_annotated.json',** snippet below**:**


```
    {
        "url": "https://4-chairs.com/products/mason-chair",
        "text_content": "mason chair regular price sale beautiful mason chair work setting contemporary casual even formal made beautiful black
leather natural wood leather chairs perfect surrounding dining table black leather wood wide deep high seat height arm height weeks returnable
store credit within days delivery date information please see return policy questions order please email orderschairscom submit close search
expand collapse expand collapse expand collapse expand collapse expand collapse expand collapse expand collapse expand collapse expand collapse
expand collapse expand collapse expand collapse expand collapse search submit log cart cart expandcollapse previous slide next slide regular
price sale add cart share share facebook tweet tweet twitter pin pin pinterest subscribe facebook pinterest instagram",
        "products": [
            {
                "name": "chair",
                "annotation": "B-Product"
            },
            {
                "name": "chair",
                "annotation": "B-Product"
            },
            {
                "name": "chairs",
                "annotation": "B-Product"
            },
            {
                "name": "dining table",
                "annotation": "B-Product, I-Product"
            },
            {
                "name": "seat",
                "annotation": "B-Product"
            }
        ]
    }
```

The last function I needed is **generate_tags (text_content, products).** This function splits text_content into words, then for each word it checks if it matches a product name from products list. If there is a match then tags are generated base on that product annotation (B-Product first word and I-Product the rest). Howeve if there is no match found then it just assigns "O". The generated tags are returned as a list. The output is then saved in another json file called **'data/annotated_output.json':**


```
    {
        "url": "https://www.hudsonfurniture.com.au/products/string-weave-timber-stool",
        "text_content": "string weave timber stool related products mcha new string weave stool made sungkai wood super comfy lightweight perfect
fit kitchen bench dimensions need know width cm depth cm height cm sign save subscribe get special offers free giveaways onceinalifetime deals
hudson furniture close menu instagram facebook twitter pinterest close cart site navigation log cart search close esc string weave timber stool
regular price add cart share share facebook tweet tweet twitter pin pin pinterest quick view quick view quick view quick view quick view quick
view close esc close esc close esc close esc close esc close esc subscribe instagram facebook twitter pinterest previous next close esc",
        "tags": [
            "O",
            "O",
            "O",
            "B-Product",
            "O",
            "O",
            "O",
            "O",
            "O",
            "O",
            "B-Product",
```


III. Train test split

I have used sklearn- train_test_split in the notebook **train_test_split.ipynb** to create two files for train and test that will be used in the NER model. I filtered the urls with no text_content as some of the url gave no error but also there was no text retrieved. Text content was first tokenized and the words are added under a new dictionary key called "words" and the text_content is deleted, as its no more needed. I split the data with 90% in train and 10% in test.

The formatted data for training and testing are saved into separate JSON files (**train_data.json and test_data.json**, respectively)

IV. NER – Model Configuration

For the NER model I have mostly kept the format from huggingface. The model is in notebook **ner.ipynb.**

I created a new folder called **output_dir_anaconda,** where all checkpoints from different NER runs are stored. Inside **output_dir_anaconda there** is another folder called **best_model**, where the weights from the best NER-Model are stored and used after in the predictions. To create unique checkpoints I have used the datetime module:

```
timestamp = datetime.datetime.now().strftime("%Y%m%d%H%M%S")  # Generate a timestamp
output_dir = os.path.join("./output_dir_anaconda", f"test-{task}-{timestamp}")  # Created unique checkpoint output directory
output_dir_best_model = "./output_dir_anaconda/best_model"
```

I wanted to test with different parameters so I created the following search space:

```
search_space = {
    "learning_rate": [1e-04, 1e-05],
    "per_device_train_batch_size": [8, 16],
    "per_device_eval_batch_size": [8, 16],
    "num_train_epochs": [25],
    "weight_decay": [1e-04, 1e-05],
}
```

```
args = TrainingArguments(
    output_dir=output_dir,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    learning_rate=config["learning_rate"],
    per_device_train_batch_size=config["per_device_train_batch_size"],
    per_device_eval_batch_size=config["per_device_eval_batch_size"],
    num_train_epochs=config["num_train_epochs"],
    weight_decay=config["weight_decay"],
    load_best_model_at_end=True,
    metric_for_best_model="f1",
    greater_is_better=True,
    # optim="adamw_torch",
     logging_strategy="epoch",
)
```

I used f1 as the best metric for my models and greater_is_better= True, the evaluation is done after each epoch. The gridsearch defines a search space consisting of different combinations of hyperparameters. It then iterates through each combination, trains the model, evaluates its performance and records the results for each configuration. The best configuration is identified based on the best f1-score.

I also used EarlyStoppingCallback with early_stopping_patience=3 to speed up the training process a bit.

Some results with the search space defined:

```
Starting training with config: {'learning_rate': 0.0001, 'per_device_train_batch_size': 8, 'per_device_eval_batch_size': 8, 'num_train_epochs': 25, 'weight_decay': 1e-05}
```

Epoch	Training Loss	Validation Loss	Precision	Recall	F1	Accuracy
1	0.303800	0.246101	0.342105	0.726708	0.465209	0.885670
2	0.124900	0.128807	0.572917	0.683230	0.623229	0.952406
3	0.060600	0.145120	0.516667	0.770186	0.618454	0.945163
4	0.042000	0.118909	0.583333	0.739130	0.652055	0.955510
5	0.025700	0.078641	0.802395	0.832298	0.817073	0.974651
6	0.013000	0.069527	0.848485	0.869565	0.858896	0.981893
7	0.006600	0.081870	0.797814	0.906832	0.848837	0.979824
8	0.006700	0.084859	0.827778	0.925466	0.873900	0.980341
9	0.005800	0.075352	0.851190	0.888199	0.869301	0.983445
10	0.003900	0.103868	0.784211	0.925466	0.849003	0.975685
11	0.003000	0.082693	0.874214	0.863354	0.868750	0.981893

```
Starting training with config: {'learning_rate': 0.0001, 'per_device_train_batch_size': 8, 'per_device_eval_batch_size': 8, 'nu
m_train_epochs': 25, 'weight_decay': 0.0001}
```

Epoch	Training Loss	Validation Loss	Precision	Recall	F1	Accuracy
1	0.323600	0.219440	0.382253	0.695652	0.493392	0.902742
2	0.131500	0.133584	0.631579	0.596273	0.613419	0.954475
3	0.063500	0.126200	0.540541	0.745342	0.626632	0.946198
4	0.042600	0.100706	0.732240	0.832298	0.779070	0.966891
5	0.032900	0.073436	0.792683	0.807453	0.800000	0.973099
6	0.016100	0.069942	0.788235	0.832298	0.809668	0.976203
7	0.008900	0.073963	0.762162	0.875776	0.815029	0.977237
8	0.009000	0.075435	0.841176	0.888199	0.864048	0.983445
9	0.006000	0.066814	0.874251	0.906832	0.890244	0.983963
10	0.002900	0.067845	0.901235	0.906832	0.904025	0.985515
11	0.002600	0.070454	0.883436	0.894410	0.888889	0.984997
12	0.001900	0.076735	0.856322	0.925466	0.889552	0.982411
13	0.001600	0.079978	0.872727	0.894410	0.883436	0.982928

```
Starting training with config: {'learning_rate': 0.0001, 'per_device_train_batch_size': 8, 'per_device_eval_batch_size': 16, 'n
um_train_epochs': 25, 'weight_decay': 1e-05}
```

Epoch	Training Loss	Validation Loss	Precision	Recall	F1	Accuracy
1	0.387700	0.230567	0.000000	0.000000	0.000000	0.915966
2	0.145600	0.149605	0.489865	0.674419	0.567515	0.941176
3	0.111000	0.105460	0.668203	0.674419	0.671296	0.960000
4	0.064200	0.087994	0.746667	0.781395	0.763636	0.969748
5	0.035400	0.076500	0.761905	0.893023	0.822270	0.974454
6	0.022900	0.064194	0.820833	0.916279	0.865934	0.981513
7	0.013000	0.062077	0.893519	0.897674	0.895592	0.986218
8	0.010700	0.060290	0.872247	0.920930	0.895928	0.985882
9	0.004600	0.057169	0.881057	0.930233	0.904977	0.987563
10	0.002900	0.061621	0.877193	0.930233	0.902935	0.986555
11	0.002200	0.063705	0.896861	0.930233	0.913242	0.987227
12	0.003300	0.065625	0.891403	0.916279	0.903670	0.987899
13	0.005900	0.064951	0.911111	0.953488	0.931818	0.989244
14	0.003400	0.061404	0.909091	0.930233	0.919540	0.989244
15	0.002200	0.064188	0.888889	0.930233	0.909091	0.987227
16	0.001700	0.065422	0.900452	0.925581	0.912844	0.987563

```
Starting training with config: {'learning_rate': 0.0001, 'per_device_train_batch_size': 8, 'per_device_eval_batch_size': 16, 'n
um_train_epochs': 25, 'weight_decay': 0.0001}
```

Epoch	Training Loss	Validation Loss	Precision	Recall	F1	Accuracy
1	0.415200	0.251368	0.000000	0.000000	0.000000	0.915966
2	0.169600	0.146590	0.529210	0.716279	0.608696	0.946555
3	0.113300	0.126573	0.598901	0.506977	0.549118	0.952605
4	0.065400	0.089282	0.788462	0.762791	0.775414	0.973782
5	0.036000	0.070575	0.844749	0.860465	0.852535	0.980504
6	0.024400	0.098981	0.687500	0.920930	0.787276	0.968067
7	0.017100	0.069901	0.900000	0.879070	0.889412	0.984202
8	0.013400	0.072076	0.860262	0.916279	0.887387	0.983193
9	0.006400	0.075080	0.872727	0.893023	0.882759	0.984202
10	0.004400	0.077219	0.882353	0.906977	0.894495	0.984874
11	0.002700	0.073771	0.895455	0.916279	0.905747	0.985882
12	0.002300	0.076541	0.865217	0.925581	0.894382	0.984202
13	0.002800	0.074893	0.900452	0.925581	0.912844	0.986891
14	0.002700	0.080175	0.871681	0.916279	0.893424	0.985210
15	0.001800	0.081314	0.847458	0.930233	0.886918	0.983193
16	0.001000	0.079953	0.885463	0.934884	0.909502	0.985882

```
Starting training with config: {'learning_rate': 0.0001, 'per_device_train_batch_size': 16, 'per_device_eval_batch_size': 8, 'n
um_train_epochs': 25, 'weight_decay': 0.0001}
```

Epoch	Training Loss	Validation Loss	Precision	Recall	F1	Accuracy
1	0.714200	0.385398	0.000000	0.000000	0.000000	0.901707
2	0.322700	0.347420	0.000000	0.000000	0.000000	0.901707
3	0.267100	0.465337	0.218147	0.701863	0.332842	0.789446
4	0.241500	0.224631	0.360000	0.167702	0.228814	0.913088
5	0.167000	0.197298	0.560000	0.521739	0.540193	0.939472
6	0.119900	0.159077	0.560606	0.689441	0.618384	0.942576
7	0.083000	0.161192	0.556650	0.701863	0.620879	0.949819
8	0.063400	0.138774	0.561321	0.739130	0.638070	0.952406
9	0.051000	0.133364	0.612022	0.695652	0.651163	0.957062
10	0.041900	0.127143	0.631868	0.714286	0.670554	0.962752
11	0.035700	0.130716	0.711111	0.795031	0.750733	0.963270
12	0.029600	0.133497	0.666667	0.732919	0.698225	0.965339
13	0.024500	0.124227	0.650273	0.739130	0.691860	0.965339
14	0.023800	0.123072	0.683616	0.751553	0.715976	0.969477

```
Starting training with config: {'learning_rate': 1e-05, 'per_device_train_batch_size': 16, 'per_device_eval_batch_size': 8, 'nu
m_train_epochs': 25, 'weight_decay': 0.0001}
```

Epoch	Training Loss	Validation Loss	Precision	Recall	F1	Accuracy
1	0.772700	0.449538	0.000000	0.000000	0.000000	0.901707
2	0.378600	0.378977	0.000000	0.000000	0.000000	0.901707
3	0.314000	0.344606	0.000000	0.000000	0.000000	0.901707
4	0.285700	0.319481	0.000000	0.000000	0.000000	0.901707

```
Starting training with config: {'learning_rate': 1e-05, 'per_device_train_batch_size': 16, 'per_device_eval_batch_size': 16, 'n
um_train_epochs': 25, 'weight_decay': 1e-05}
```

Epoch	Training Loss	Validation Loss	Precision	Recall	F1	Accuracy
1	0.772700	0.449538	0.000000	0.000000	0.000000	0.901707
2	0.378600	0.378977	0.000000	0.000000	0.000000	0.901707
3	0.314000	0.344606	0.000000	0.000000	0.000000	0.901707
4	0.285700	0.319481	0.000000	0.000000	0.000000	0.901707

```
Starting training with config: {'learning_rate': 1e-05, 'per_device_train_batch_size': 16, 'per_device_eval_batch_size': 16, 'n
um_train_epochs': 25, 'weight_decay': 0.0001}
```

Epoch	Training Loss	Validation Loss	Precision	Recall	F1	Accuracy
1	0.772700	0.449538	0.000000	0.000000	0.000000	0.901707
2	0.378600	0.378977	0.000000	0.000000	0.000000	0.901707
3	0.314000	0.344606	0.000000	0.000000	0.000000	0.901707
4	0.285700	0.319481	0.000000	0.000000	0.000000	0.901707

BEST RESULTS :

```
Best hyperparameters found: {'learning_rate': 0.0001, 'per_device_train_batch_size': 8, 'per_device_eval_batch_size': 16, 'num_
train_epochs': 25, 'weight_decay': 1e-05}
Best F1 score: 0.9318181818181819
```

Results with lemmatization (I have not tested with multiple hyperparameters, only with the best ones found above):

Epoch	Training Loss	Validation Loss	Precision	Recall	F1	Accuracy
1	0.417900	0.327977	0.000000	0.000000	0.000000	0.908277
2	0.245500	0.257934	0.564516	0.315315	0.404624	0.922446
3	0.117900	0.140508	0.735537	0.801802	0.767241	0.967189
4	0.061700	0.107599	0.762295	0.837838	0.798283	0.970172
5	0.043400	0.107363	0.767241	0.801802	0.784141	0.973154
6	0.035500	0.091954	0.758065	0.846847	0.800000	0.974646
7	0.027100	0.109053	0.887755	0.783784	0.832536	0.973900
8	0.020700	0.095572	0.821429	0.828829	0.825112	0.979120
9	0.011000	0.095787	0.879630	0.855856	0.867580	0.979120
10	0.008400	0.102241	0.837838	0.837838	0.837838	0.977629
11	0.004800	0.101631	0.810345	0.846847	0.828194	0.976883
12	0.003300	0.118262	0.833333	0.855856	0.844444	0.977629
13	0.002800	0.113738	0.850877	0.873874	0.862222	0.976883
14	0.002400	0.098609	0.860870	0.891892	0.876106	0.979120
15	0.003300	0.082780	0.859649	0.882883	0.871111	0.982103
16	0.002100	0.087946	0.875000	0.882883	0.878924	0.981357
17	0.001600	0.083653	0.862069	0.900901	0.881057	0.980611
18	0.001100	0.089057	0.858407	0.873874	0.866071	0.979120
19	0.001100	0.094053	0.891892	0.891892	0.891892	0.981357
20	0.000600	0.093803	0.881818	0.873874	0.877828	0.980611
21	0.000600	0.092375	0.873874	0.873874	0.873874	0.979866
22	0.000900	0.088714	0.873874	0.873874	0.873874	0.979866
23	0.000900	0.085519	0.866071	0.873874	0.869955	0.980611
24	0.000500	0.085776	0.883929	0.891892	0.887892	0.980611

For predictions we load the tokenizer and model from the "./output_dir_anaconda/best_model". The dictionary id2tag is created to map the predicted labels to their corresponding tags. To prepare the data for prediction I have run the scrapy spider again but this time for url from position 200 to the end (as first 200 urls were used for training), the data is stored in **data/to_predict.json**. The function **predict_for_urls** reads text data and URL pairs from the JSON file from **data/to_predict.json**, segments the text into chunks based on a maximum length parameter, and then utilizes the **predict** function to make predictions for each segment. As for some urls the text_content was too large I needed to split the data in smaller schunks as when I first ran the code as it is I got a warning stating that the length is longer than the max_length accepted and the predictions might be wrong. I have checked and indeed when the text_content was too long I got many incorrect predictions.

The **predict** function tokenizes input sentences, encodes them using the tokenizer, and feeds them to the model to obtain predictions. These predictions are then decoded using the **id2tag** mapping to interpret the predicted labels, and non-entity tokens are filtered out.

Next, in the same notebook, I have concatenated the words split by chars " ##". If a word starts with "##" then it appends it to the previous word.
Before:


```
{'url': 'https://asianteakfurniture.com/products/bali-teak-bench-atf388',
'predictions': [('bench', 'B-Product'), ('bench', 'B-Product'), ('fur', 'B
-Product'), ('##nish', 'B-Product'), ('##ing', 'B-Product'), ('bench', 'B-
Product')]}
```

After:


```
{'url': 'https://asianteakfurniture.com/products/bali-teak-bench-atf388',
'predictions': [('bench', 'B-Product'), ('bench', 'B-Product'), ('furnishi
ng', 'B-Product'), ('bench', 'B-Product')]}
```


Then I also wanted to concatenated the groups of words formed from B-Product I-Product. Meaning that if a word is tagged I-Product it combines it to the previous word for a multi_word concatenation, the tag I-Product also gets concatenated with the previous word's tag.

Before:


```
{'url': 'https://furnitica-vinova.myshopify.com/products/enim-donec-pede',
'predictions': [('cupboard', 'B-Product'), ('dinner', 'B-Product'), ('tabl
e', 'I-Product'), ('table', 'B-Product'), ('sofa', 'B-Product'), ('clock',
'B-Product'), ('foot', 'B-Product'), ('table', 'B-Product'), ('fireplace',
'B-Product'), ('table', 'B-Product'), ('lamp', 'I-Product'), ('ottoman', '
B-Product'), ('armchair', 'B-Product'), ('cushion', 'B-Product'), ('coffee
', 'B-Product'), ('table', 'I-Product'), ('shelf', 'B-Product'), ('sofa',
'B-Product'), ('table', 'B-Product'), ('curtain', 'B-Product'), ('chandeli
er', 'B-Product'), ('wardrobe', 'B-Product'), ('floor', 'B-Product'), ('la
mp', 'I-Product'), ('bed', 'B-Product'), ('bedside', 'B-Product'), ('table
', 'I-Product')]}
```

After:


```
{'url': 'https://furnitica-vinova.myshopify.com/products/enim-donec-pede',
'predictions': [('cupboard', 'B-Product'), ('dinner table', 'B-Product I-P
roduct'), ('table', 'B-Product'), ('sofa', 'B-Product'), ('clock', 'B-Prod
uct'), ('foot', 'B-Product'), ('table', 'B-Product'), ('fireplace', 'B-Pro
duct'), ('table lamp', 'B-Product I-Product'), ('ottoman', 'B-Product'), (
'armchair', 'B-Product'), ('cushion', 'B-Product'), ('coffee table', 'B-Pr
oduct I-Product'), ('shelf', 'B-Product'), ('sofa', 'B-Product'), ('table'
, 'B-Product'), ('curtain', 'B-Product'), ('chandelier', 'B-Product'), ('w
ardrobe', 'B-Product'), ('floor lamp', 'B-Product I-Product'), ('bed', 'B-
Product'), ('bedside table', 'B-Product I-Product')]}
```

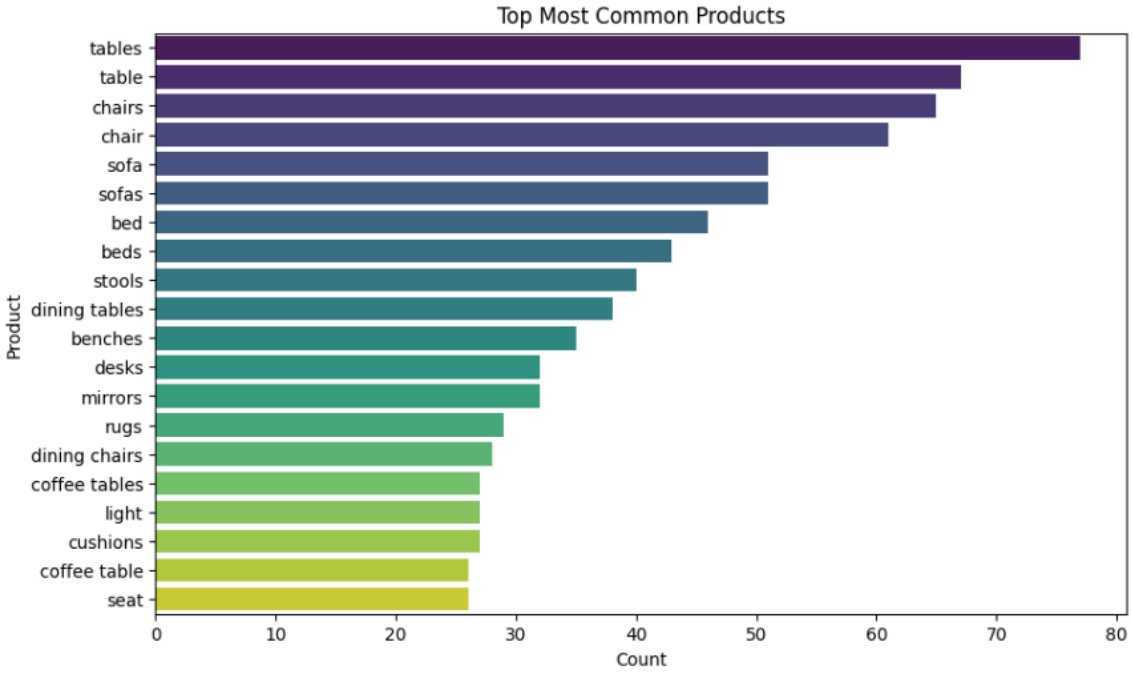

Finally in a new json called **./data/final_predictions.json** I saved the urls along with the predicted words.


```
[
    {
        "url": "https://www.myconcept.com.hk/products/moo",
        "predictions": [
            "light",
            "lamp",
            "light",
            "bulbs",
            "light",
            "lamp",
            "lamp",
            "lamp"
        ]
    },
```

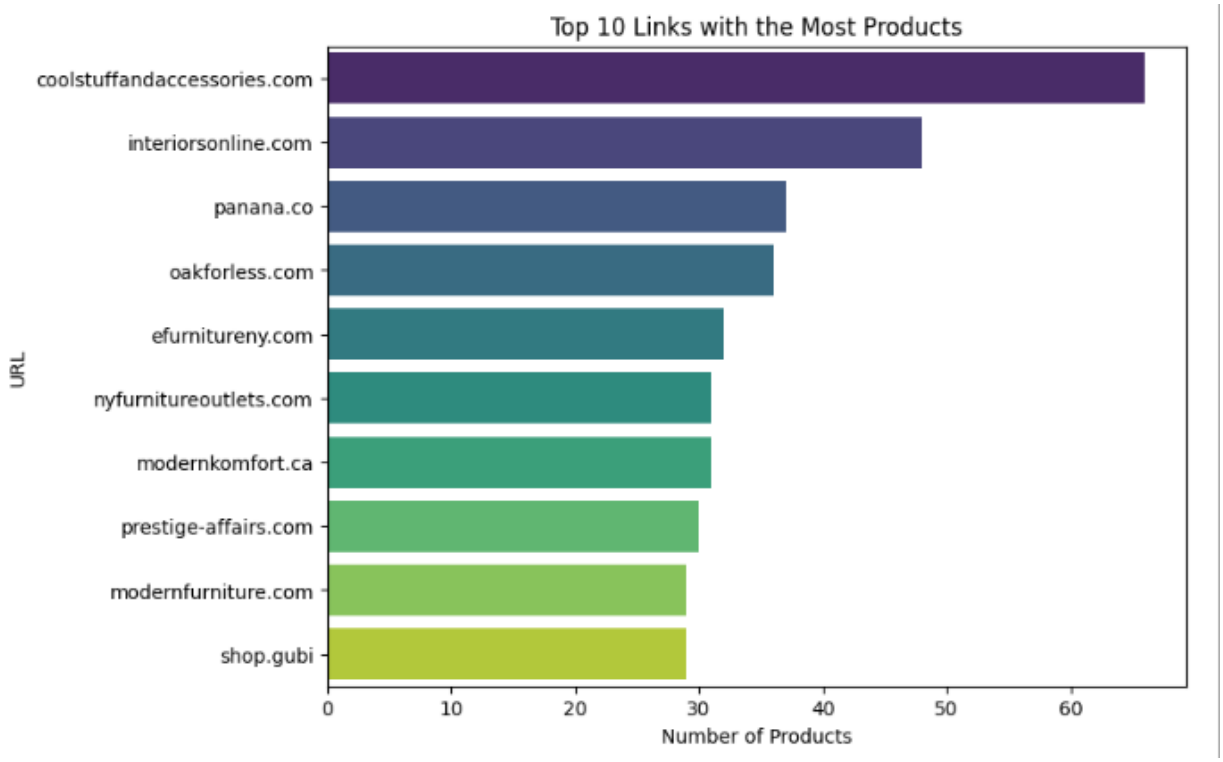

V. Data concatenation and vizualization

In the notebook save_and_visualize_data.ipynb I concatenated the data from training from the json file **"./data/produse_in_urls.json"** with the data from the predicitons file **"./data/final_predictions.json",** and added them in single pandas dataframe which I later saved under a csv in **'./data/full_list_products'.** Also for both train and test data I have kept only one entry of a product per url, so the same product does not appear multiple times in the final file.

Finally I created some data visualization to see top 20 most frequent furniture products that appear:



And also wanted to see which stores have the most number of different products and also print what those are:



URL: coolstuffandaccessories.com
Products: ['headboards', 'headphones', 'table', 'seats', 'cabinet', 'books
helf', 'coffee tables', 'dining table', 'dining chair sofa', 'shelf', 'bar
stool', 'tables', 'bookcase', 'bench', 'office chair', 'ottomans', 'case',
'ottoman', 'sleeper', 'nightstands', 'islands', 'chandelier', 'desk', 'dre
ssers table lamps', 'shelves', 'sofas', 'dining chair', 'book', 'bookcases
', 'chests', 'futons', 'pillow', 'stools table', 'buffet', 'sectional sofa
units', 'beds', 'chaise loungers', 'console', 'recliners', 'stools', 'chai
r', 'console table', 'love seat', 'bed', 'stool', 'benches', 'office chair
s', 'buffets', 'arm', 'bar', 'dresser', 'cabinets', 'lounge chair', 'chest
', 'mirror', 'coffee table', 'mirrors', 'sofa', 'deskphones', 'armchair st
ool', 'lounge', 'nightstand', 'patio chairs', 'desks', 'chaise', 'chairs']

URL: interiorsonline.com
Products: ['bedside tables', 'headboards', 'table', 'seats', 'gift cards',
'gift card', 'coffee tables', 'dining table', 'double beds', 'lounge chair
s', 'lights', 'tables', 'rugs', 'bench', 'ottomans', 'floor lamps', 'rug',
'shelves', 'sofas', 'work tables', 'desk lamps', 'lamps', 'dining chairs',
'bookcases', 'chests', 'armchair', 'couches', 'beds', 'office desks', 'sto
ols', 'bed', 'benches', 'chest drawers', 'office chairs', 'buffets', 'desk
s', 'mirrors', 'sofa', 'console tables', 'cabinets', 'table lamps', 'cushi
ons', 'bookshelves', 'drawer', 'armchairs', 'dining tables', 'bar stools',
'chairs']

URL: panana.co
Products: ['bedside cabinets', 'table', 'sideboard', 'tub chairs', 'dining
table', 'tables', 'bookcase', 'ottoman', 'desk chair', 'desk', 'shelves',
'bench chairs', 'recliner', 'dining chairs', 'barber chairs', 'console tab
le', 'stools', 'chair', 'massage chair', 'stool', 'bed', 'chest drawers',
'mattress', 'dressing table', 'accent chairs', 'bar', 'sofa', 'cabinets',
'sofa bed', 'sofa table table', 'coffee table', 'end tables', 'dining tabl
e sofa table table', 'dining table table', 'footstool', 'bedside', 'chairs
']

URL: oakforless.com
Products: ['headboards', 'gift cards', 'gift card', 'hutches', 'cupboards'
, 'tables', 'bookcase', 'writing desks', 'arm chairs', 'kitchen islands',
'nightstands', 'hall trees', 'curios', 'mattresses', 'sofas', 'pantry', 'd
ining chairs', 'file cabinets', 'chests', 'bookcases', 'barstools', 'beds'
, 'office desks', 'side chairs', 'stools', 'dressers', 'benches', 'office
chairs', 'drawers', 'buffets', 'sofa', 'cabinets', 'end tables', 'light',
'desks', 'dining tables']

URL: efurnitureny.com
Products: ['office chair', 'tables', 'rugs', 'bunk beds', 'ottomans', 'isl
ands', 'curios', 'mattresses', 'vanity tables', 'bar chairs', 'dining chai
rs', 'bookcases', 'dresser mirrors', 'beds', 'recliners', 'love seat', 'ch
air', 'benches', 'buffets', 'mattress', 'accent chairs', 'mirrors', 'cabin
ets', 'sofa', 'chest', 'console tables', 'coffee table', 'night stands', '
cribs', 'desks', 'dining tables', 'wardrobe']

URL: nyfurnitureoutlets.com

Products: ['headboard', 'sofa beds', 'tables', 'rugs', 'bunk beds', 'night stands', 'filing cabinets', 'sofas', 'recliner', 'dining chairs', 'bookcases', 'chests', 'beds', 'recliners', 'stools', 'dressers', 'bed', 'benches', 'pillows', 'office chairs', 'buffets', 'arm', 'mattress', 'dining chairs tables tables', 'bar', 'mirrors', 'patio chairs ottomans', 'fireplaces', 'counter', 'desks', 'chairs']

URL: modernkomfort.ca
Products: ['table', 'coffee tables', 'lounge chairs', 'tables', 'rugs', 'ottomans', 'floor lamps', 'kitchen islands', 'nightstands', 'vanity', 'rug', 'sofas', 'desk lamps', 'lamps', 'dining chairs', 'bookcases', 'beds', 'clocks', 'stools', 'dressers', 'bed', 'benches', 'pillows', 'office chairs', 'mirror', 'console tables', 'cabinets', 'mirrors', 'table lamps', 'desks', 'dining tables']

URL: prestige-affairs.com
Products: ['seat', 'table', 'coffee tables', 'dining table', 'lounge chairs', 'shelf', 'tables', 'rugs', 'arm chairs', 'ottoman', 'rug', 'shelves', 'sofas', 'sectional sofa', 'dining chair', 'dining chairs', 'console', 'stools', 'chair', 'bed', 'benches', 'consoles', 'mattress', 'bar', 'mirrors', 'sofa', 'light', 'bedside', 'dining tables', 'chairs']

URL: modernfurniture.com
Products: ['bedside tables', 'office chairs cabinets', 'office', 'dining tables', 'coffee tables', 'lounge chairs', 'reception', 'cupboards', 'sofa beds', 'tables', 'rugs', 'ottomans', 'sofas', 'lamps', 'dining chairs', 'beds', 'stools', 'dressers', 'benches', 'office chairs', 'desks', 'console tables', 'sofa', 'cabinets', 'cushions', 'bookshelves', 'armchairs', 'chaise', 'chairs']

URL: shop.gubi
Products: ['coffee tables', 'lounge chairs', 'tables', 'rugs', 'ottomans', 'floor lamps', 'lamp lamp', 'patio desks', 'sofas', 'dining chair', 'lounge tables', 'lamp', 'lamps', 'dining chairs', 'mirror lamp', 'dining chair lamp', 'stools', 'chair', 'consoles', 'bar', 'mirrors', 'lounge chair', 'table lamps', 'coffee table', 'table lamp', 'patio mirrors', 'desks', 'dining tables', 'chairs']