```
#%% md
# Import Functions
#%%
import pandas as pd
from tqdm import tqdm
import random as rd
import torch
from torch.utils.data import Dataset, DataLoader
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import classification_report
#%% md
# Initiate lists of data to load into training
#%% md
## Product Types list and query funciton
#%%
def product_list_creation(file_name: str) -> list:
    products_df = pd.read_excel(file_name, sheet_name='product_type')

    unique_family = products_df['Family'].dropna().unique().tolist()
    product_types = products_df['ProductType'].dropna().tolist()
    keywords = [i.split(', ') for i in products_df['Keywords'].dropna().tolist()]
    keyword_list = list(set([w for word in keywords for w in word]))

    product_types.extend(keyword_list)
    product_types.extend(unique_family)
    product_types = list(set(product_types))
    return product_types
#%%
def product_query(product_types):
    queries = []
    labels = []
    for word in product_types:
        labels_product = ['B-Product'] + ['I-Product'
```

```
36 ] * (len(word.split(' ')) - 1)
37         queries.append(word.split(' '))
38         labels.append(labels_product)
39     return queries, labels
40 #%% md
41 ## Material Types list and query function
42 #%%
43 def material_list_creation(file_name: str) -> list:
44     materials_df = pd.read_excel(file_name,
   sheet_name='material_type')
45 
46     material_types = materials_df['material_type'].
   dropna().tolist()
47     mat_fam = materials_df['family'].dropna().tolist
   ()
48     mat_k_word = list(set([w for words in [i.split(
   ', ') for i in materials_df['keywords'].dropna().
   tolist()] for w in words]))
49 
50     material_types.extend(mat_fam)
51     material_types.extend(mat_k_word)
52     material_types = list(set(material_types))
53     return material_types
54 #%%
55 def material_query(material_types):
56     queries = []
57     labels = []
58     for word in material_types:
59         word = word.lower()
60         labels_material = ['B-Material'] + ['I-
   Material'] * (len(word.split(' ')) - 1)
61         queries.append(word.split(' '))
62         labels.append(labels_material)
63     return queries, labels
64 #%% md
65 ## Building Applications list and query function
66 #%%
67 def building_app_list_creation(file_name: str) ->
   list:
68     building_app_df = pd.read_excel(file_name,
   sheet_name='building_applications')
```

```
 69
 70     building_app_list = building_app_df['building_applications'].dropna().tolist()
 71     keywords = list(set([w for words in [i.split(', ') for i in building_app_df['keywords'].dropna().tolist()] for w in words]))
 72     building_app_list.extend(keywords)
 73     return building_app_list
 74 #%%
 75 def building_app_query(building_app_list):
 76     queries = []
 77     labels = []
 78     for word in building_app_list:
 79         labels_building_application = ['B-Application'] + ['I-Application'] * (len(word.split(' ')) - 1)
 80         queries.append(word.split(' '))
 81         labels.append(labels_building_application)
 82     return queries, labels
 83 #%% md
 84 ## Country list and query function
 85 #%%
 86 def country_list_creation(file_name: str) -> list:
 87     country_df = pd.read_excel(file_name, sheet_name='countries')
 88     country_list = country_df['country'].dropna().tolist()
 89     return country_list
 90 #%%
 91 def country_query(country_list):
 92     queries = []
 93     labels = []
 94     for word in country_list:
 95         labels_country = ['B-Country'] + ['I-Country'] * (len(word.split(' ')) - 1)
 96         queries.append(word.split(' '))
 97         labels.append(labels_country)
 98     return queries, labels
 99 #%% md
100 ## Recycled Content list and query
101 #%%
```

```
def recycled_list_creation(file_name: str) -> list:
    recycled_content_df = pd.read_excel(file_name, sheet_name='recycled_content')
    recycled_list = recycled_content_df['recycled_content'].dropna().str.lower().tolist()
    recycle_keywords = list(set([w for words in [i.split(', ') for i in recycled_content_df['keywords'].dropna().tolist()] for w in words]))
    recycled_list.extend(recycle_keywords)

    final_list = []
    for item in recycled_list:
        if 'x%' in item:
            for _ in range(21):
                final_list.append(item.replace('x%', str(rd.randint(0, 100)) + '%'))
        elif '.x' in item:
            for _ in range(21):
                final_list.append(item.replace('.x', '.' + str(rd.randint(0, 100))))
        else:
            final_list.append(item)
    return final_list
#%%
def recycle_query(recycled_list):
    queries = []
    labels = []
    for word in recycled_list:
        labels_country = ['B-Recycle'] + ['I-Recycle'] * (len(word.split(' ')) - 1)
        queries.append(word.split(' '))
        labels.append(labels_country)
    return queries, labels
#%%
'''
EXCEL FILE WITH SHEETS OF UNIQUE PRODUCT/MATERIAL/BUILDINGAPPLICATION TYPE REDACTED
PROPRIETARY KNOWLEDGE OF 2050 MATERIALS
CONTACT info@2050-materials.com FOR MORE INFORMATION
'''
def list_initialization(file_name: str ='
```

```
YOUR_EXCEL_FILE.xlsx'):
    product_types = product_list_creation(file_name)
    material_types = material_list_creation(file_name)
    building_app_list = building_app_list_creation(file_name)
    country_list = country_list_creation(file_name)
    recycled_list = recycled_list_creation(file_name)

    return product_types, material_types, building_app_list, country_list, recycled_list
#%% md
## Creates query with proper label for every word in  product types, material types, building applications, countries, and recycled content
#%%
def all_queries(product_types, material_types, building_app_list, country_list, recycled_list):
    queries = []
    labels = []

    p_queries, p_labels = product_query(product_types)
    queries.extend(p_queries)
    labels.extend(p_labels)

    m_queries, m_labels = material_query(material_types)
    queries.extend(m_queries)
    labels.extend(m_labels)

    b_queries, b_labels = building_app_query(building_app_list)
    queries.extend(b_queries)
    labels.extend(b_labels)

    c_queries, c_labels = country_query(country_list)
    queries.extend(c_queries)
    labels.extend(c_labels)
```

```
164 
165     r_queries, r_labels = recycle_query(
    recycled_list)
166     queries.extend(r_queries)
167     labels.extend(r_labels)
168 
169     return queries, labels
170 #%% md
171 ## Creation of template sentences, all words in
    database added with "all_words='y'"
172 #%%
173 def new_sentences(num, product_types, material_types
    , building_app_list, country_list, recycled_list,
    all_words='n'):
174     # template sentences that selects information at
     random to be selected for training data
175     templates = [
176                 "{product_type} products
    manufactured {country} for {building_application}",
177                 "{material_type} manufactured in {
    country} for {building_application} with {
    recycled_content} recycled content",
178                 "{product_type} manufactured {
    country} with {recycled_content} recycled content",
179                 "{product_type} made in {country}
    for {building_application} with {recycled_content}
    recycle content",
180                 "Find {product_type} from {country
    } used in {building_application}",
181                 "Do you have any {product_type}
    used in {building_application}?",
182                 "I'm looking for {material_type}
    products from {country} for {building_application}."
    ,
183                 "{product_type} from {country}
    suitable for {building_application}?",
184                 "Search for {material_type}
    materials from {country} used in {
    building_application} applications",
185                 "Can I find {product_type} suitable
     for {building_application} applications?",
```

```
                  "What types of {material_type} from {country} are used in {building_application}?",
                  "Show me {product_type} suitable for {building_application}",
                  "List all {material_type} from {country} used in {building_application}.",
                  "Are there any {product_type} from {country} for {building_application}?",
                  "Find {product_type} with {recycled_content} recycled content.",
                  "Find {material_type} with {recycled_content} recycled content.",
                  "List all {material_type} from {country} with recycled content {recycled_content} "
                  ]

    queries = []
    labels = []

    for null in range(num):
        temp = rd.choice(templates)
        material_type = rd.choice(material_types)
        product_type = rd.choice(product_types)
        building_application = rd.choice(building_app_list)
        country = rd.choice(country_list)
        recycled_content = rd.choice(recycled_list)
        sentence = temp.format(
                                material_type=material_type,
                                product_type=product_type,
                                building_application=building_application,
                                country=country,
                                recycled_content=recycled_content
                                )

        queries.append(sentence.lower())
        # Generate labels
```

```
215         labels_product = ['B-Product'] + ['I-Product
    '] * (len(product_type.split(' ')) - 1)
216         labels_material = ['B-Material'] + ['I-
    Material'] * (len(material_type.split(' ')) - 1)
217         labels_country = ['B-Country'] + ['I-Country
    '] * (len(country.split(' ')) - 1)
218         labels_building_application = ['B-
    Application'] + ['I-Application'] * (len(
    building_application.split(' ')) - 1)
219         labels_recycled_content = ['B-Recycle'] + ['
    I-Recycle'] * (len(str(recycled_content).split(' '
    )) - 1)
220 
221         # Identify the placeholders in the template
    and replace them with the corresponding labels
222         template_labels = temp.split()
223         new_template_labels = []
224         for word in template_labels:
225             if 'product_type' in word:
226                 new_template_labels.extend(
    labels_product)
227             elif 'material_type' in word:
228                 new_template_labels.extend(
    labels_material)
229             elif 'country' in word:
230                 new_template_labels.extend(
    labels_country)
231             elif 'building_application' in word:
232                 new_template_labels.extend(
    labels_building_application)
233             elif 'recycled_content' in word:
234                 new_template_labels.extend(
    labels_recycled_content)
235             else:
236                 new_template_labels.append('O')
237 
238         template_labels = new_template_labels
239         labels.append(template_labels)
240 
241     if all_words == 'y':
242         solo_queries, solo_labels = all_queries(
```

```
product_types, material_types, building_app_list, country_list, recycled_list)
        queries.extend(solo_queries)
        labels.extend(solo_labels)
        print(f'{len(solo_queries)} words from database and {num} random sentences added to queries and labels data.')
    else:
        print(f'{num} random sentences added to queries and labels data.')
    return queries, labels
#%% md
## CustomDataset class to process words for tokeninzer
#%%
class CustomDataset(Dataset):
    def __init__(self, tokenizer, sentences, labels, max_len, pad_token_label_id):
        self.len = len(sentences)
        self.sentences = sentences
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len
        self.pad_token_label_id = pad_token_label_id

    def __getitem__(self, index):
        sentence = str(self.sentences[index])
        inputs = self.tokenizer.encode_plus(
            sentence,
            None,
            add_special_tokens=True,
            max_length=self.max_len,
            padding='max_length',
            return_token_type_ids=True
        )
        ids = inputs['input_ids']
        mask = inputs['attention_mask']

        label = ['O'] + self.labels[index] + ['O']
        label = [Token_Classification_Labels.get(l) for l in label]
```

```
276         padding_length = self.max_len - len(label)
277         label.extend([self.pad_token_label_id]*
    padding_length)
278         label = label[:self.max_len]
279         return {
280             'input_ids': torch.tensor(ids, dtype=
    torch.long),
281             'attention_mask': torch.tensor(mask,
    dtype=torch.long),
282             'labels': torch.tensor(label, dtype=
    torch.long)
283         }
284 
285     def __len__(self):
286         return self.len
287 #%% md
288 ## Loads queries and labels into train dataset to
    prepare for training
289 #%%
290 def train_data_info(queries, labels, tokenizer):
291     # First, split into train+val and test sets
292     sentences_temp, sentences_test, labels_temp,
    labels_test = train_test_split(queries, labels,
    test_size=0.1, random_state=42)
293 
294     # Then split train+val into separate train and
    val sets
295     sentences_train, sentences_val, labels_train,
    labels_val = train_test_split(sentences_temp,
    labels_temp, test_size=0.1, random_state=42)
296 
297     # Padding label
298     PAD_TOKEN_LABEL_ID = -100
299 
300     # Now, prepare the training, validation, and
    test data
301     train_data = CustomDataset(tokenizer,
    sentences_train, labels_train, max_len=50,
    pad_token_label_id=PAD_TOKEN_LABEL_ID)
302     valid_data = CustomDataset(tokenizer,
    sentences_val, labels_val, max_len=50,
```

```
302 pad_token_label_id=PAD_TOKEN_LABEL_ID)
303     test_data = CustomDataset(tokenizer,
    sentences_test, labels_test, max_len=50,
    pad_token_label_id=PAD_TOKEN_LABEL_ID)
304 
305     # Initialize the data loaders
306     train_dataloader = DataLoader(train_data,
    batch_size=32, shuffle=True)
307     valid_dataloader = DataLoader(valid_data,
    batch_size=32, shuffle=True)
308     test_dataloader = DataLoader(test_data,
    batch_size=32, shuffle=False)  # we don't need to
    shuffle the test set
309     return train_dataloader, valid_dataloader,
    test_dataloader
310 #%%
311 def Initialize_tokenizer():
312     from transformers import DistilBertTokenizerFast
313     # Initialize the tokenizer
314     tokenizer = DistilBertTokenizerFast.
    from_pretrained('distilbert-base-uncased')
315     return tokenizer
316 #%%
317 def Token_Classification_Initialization():
318     from transformers import
    DistilBertForTokenClassification
319     Token_Classification_Labels = {
320                                   'O': 0,
321                                   'B-Product': 1,
322                                   'I-Product': 2,
323                                   'B-Material': 3,
324                                   'I-Material': 4,
325                                   'B-Country': 5,
326                                   'I-Country': 6,
327                                   'B-Application': 7
    ,
328                                   'I-Application': 8
    ,
329                                   'B-Recycle': 9,
330                                   'I-Recycle': 10
331                                   }
```

```
    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
    Token_Classification_model = DistilBertForTokenClassification.from_pretrained(
                                        "distilbert-base-uncased",
                                        num_labels=len(Token_Classification_Labels)).to(device)
    return Token_Classification_Labels, Token_Classification_model
#%%
def Token_Classification_Model_Training(

    Token_Classification_Labels: dict,

    Token_Classification_model,

    train_dataloader,

    test_dataloader,

    Model_load_file: None|str = None,

    learning_rate: float = 1e-5,
                                            epoch_loop : int = 5
                                            ) -> list:
    optimizer = torch.optim.AdamW(params=Token_Classification_model.parameters(), lr=learning_rate)
    # Loads model if there is provided file name
    if Model_load_file:
        Token_Classification_model.load_state_dict(torch.load(Model_load_file))
        print(f'{Model_load_file} successfully loaded.')
    # Set the device
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    Token_Classification_model.to(device)

```

```
356     Token_Classification_model.train()
357 
358     # Training loop
359     training_loss = []
360     raw_results = []
361     print('Training starting.')
362     for null in tqdm(range(epoch_loop)):  # loop
    over the dataset multiple time
363         total_loss = 0
364         for i, batch in enumerate(train_dataloader,
    0):
365             # zero the parameter gradients
366             optimizer.zero_grad()
367 
368             # get the inputs
369             inputs = batch['input_ids'].to(device)
370             labels = batch['labels'].to(device)
371             masks = batch['attention_mask'].to(
    device)
372 
373             # forward + backward + optimize
374             outputs = Token_Classification_model(
    inputs, attention_mask=masks, labels=labels)
375             loss = outputs.loss
376 
377             loss.backward()
378             optimizer.step()
379 
380             # add the loss to the list
381             total_loss += loss.item()
382 
383         Token_Classification_model.eval()
384 
385         predictions = []
386         true_labels = []
387 
388         for i, batch in enumerate(test_dataloader):
389             b_input_ids = batch['input_ids'].to(
    device)
390             b_input_mask = batch['attention_mask'].
    to(device)
```

```
391             b_labels = batch['labels'].to(device)
392 
393             with torch.no_grad():
394                 outputs = Token_Classification_model
    (b_input_ids, attention_mask=b_input_mask)
395 
396             logits = outputs[0].detach().cpu().numpy
    ()
397             label_ids = b_labels.to('cpu').numpy()
398 
399             # Calculate the predictions
400             predictions.extend([list(p) for p in np.
    argmax(logits, axis=2)])
401             true_labels.extend(label_ids)
402 
403         # Define mapping from index to labels
404         idx2label = {v: k for k, v in
    Token_Classification_Labels.items()}
405         idx2label[-100] = 'PAD'
406 
407         # Flatten the output tensors
408         predictions_flat = [item for sublist in
    predictions for item in sublist]
409         true_labels_flat = [item for sublist in
    true_labels for item in sublist]
410 
411         # Map indices back to labels
412         pred_tags_flat = [idx2label[p] for p in
    predictions_flat]
413         true_tags_flat = [idx2label[t] for t in
    true_labels_flat]
414 
415         # Filter out 'PAD' labels
416         pred_tags = [pred for pred, true in zip(
    pred_tags_flat, true_tags_flat) if true != 'PAD']
417         true_tags = [true for true in true_tags_flat
     if true != 'PAD']
418 
419         # Binarize the labels
420         lb = LabelBinarizer()
421         y_true_combined = lb.fit_transform(true_tags
```

```
421 )
422         y_pred_combined = lb.transform(pred_tags)
423 
424         # appends classification report into list
    for each epoch
425         class_report = classification_report(
    y_true_combined, y_pred_combined, target_names=lb.
    classes_, digits=4, zero_division=1, output_dict=
    True)
426         raw_results.append(class_report)
427         Token_Classification_model.train()
428 
429         # Save training loss
430         training_loss.append(total_loss/len(
    train_dataloader))
431 
432     raw_results.append(training_loss)
433     print('Training Complete.')
434     if Model_load_file:
435         torch.save(Token_Classification_model.
    state_dict(), Model_load_file)
436         print(f'{Model_load_file} successfully saved
    .')
437     else:
438         Model_load_file = 'Token_Class_Model.pth'
439         torch.save(Token_Classification_model.
    state_dict(), Model_load_file)
440         print(f'{Model_load_file} successfully saved
    .')
441     return raw_results
442 #%% md
443 ## Organizes results from training into dictionary
444 #%%
445 def results_dict_creation(raw_results: list) -> dict
    :
446     training_loss = raw_results.pop()
447     results_dict = {
448                 'epoch': [],
449                 'precision': [],
450                 'recall': [],
451                 'f1_score': [],
```

```
452                 'training loss': []
453                 }
454     for index, item in enumerate(raw_results):
455         results_dict['epoch'].append(index+1)
456         results_dict['precision'].append(item['
    weighted avg']['precision'])
457         results_dict['recall'].append(item['weighted
     avg']['recall'])
458         results_dict['f1_score'].append(item['
    weighted avg']['f1-score'])
459         results_dict['training loss'].append(
    training_loss[index])
460     return results_dict
461 #%% md
462 ## Takes Epoch df column and column to compare to
    output graph of progression over each epoch
463 #%%
464 def plot_creation(epochs, y_value):
465     import matplotlib.pyplot as plt
466 
467     epoch_nums = epochs.tolist()
468     y_val_list = y_value.tolist()
469 
470     x = epochs.values.reshape(-1,1)
471     y = y_value.values.reshape(-1,1)
472 
473     linear_r = LinearRegression()
474     linear_r.fit(x, y)
475     y_pred = linear_r.predict(x)
476     plt.plot(epoch_nums, y_pred, color='red',
    linestyle='dashdot')
477 
478     plt.plot(epoch_nums, y_val_list, color='blue')
479     plt.title(y_value.name)
480     plt.xlabel('epoch')
481     plt.legend(['linear regression',y_value.name])
482 
483     plt.tight_layout()
484     plt.show()
485 #%% md
486 ## Saves results of training into .csv file
```

```
#%%
def save_results(df):
    import datetime
    import pytz
    '''
    SAVES ALL TRAINING RESULTS, PRECISION RECALL F1_SCORE TRAINING LOSS
    INTO .csv FILE IN YEAR-MONTH-DAY_HOUR-MINUTE-SECOND.csv FILE NAME,
    LINKED WITH YOUR TIMEZONE SO NO DUPLICATE FILES AND KNOWLEDGE OF WHEN TRAINING IS COMPLETE
    '''
    current_time = datetime.datetime.now(pytz.timezone('REPLACE_WITH_YOUR_TIMEZONE'))
    save_location = 'FOLDER_LOCATION_OF_TRAINING_DATA' + current_time.strftime('%Y-%m-%d_%H-%M-%S') + '.csv'
    df.to_csv(save_location, sep=',', index=False)
    print(f'File {save_location} saved.')
#%% md
## Function to take sentance/query from user and process result, output in dictionary {str:list}
#%%
def Token_Classification_Results(List_of_Sentences: list,

    Token_Classification_Labels: dict,

    Token_Classification_model,
                                    tokenizer,
                                    Token_Class_File: str
                                    ) -> dict:
    # Puts model into eval mode
    Token_Classification_model.eval()

    # Loads the saved file of Pre-trained model
    Token_Classification_model.load_state_dict(torch.load(Token_Class_File))
    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
```

```
515 
516     # Inverse mapping from label indices to labels
517     inv_label_map = {v: k for k, v in
    Token_Classification_Labels.items()}
518 
519     # Creating Final Dictionary for results
520     final_dict = {
521                    'Product Type': [],
522                    'Material Type': [],
523                    'Building applications': [],
524                    'Countries': [],
525                    'Recycled Content': []
526                    }
527 
528     # Cycling through query
529     for sentence in List_of_Sentences:
530         sentence = sentence.lower()
531         # Encode the sentence
532         inputs = tokenizer.encode_plus(
533                                             sentence,
534                                             None,
535 
    add_special_tokens=True,
536                                             padding='
    longest',
537 
    return_token_type_ids=True
538                                             )
539 
540         # Create torch tensors and move them to the
    device
541         input_ids = torch.tensor([inputs['input_ids'
    ]], dtype=torch.long).to(device)
542         attention_mask = torch.tensor([inputs['
    attention_mask']], dtype=torch.long).to(device)
543 
544         # Run the sentence through the model
545         with torch.no_grad():
546             outputs = Token_Classification_model(
    input_ids, attention_mask=attention_mask)
547 
```

```
548         # Get the token-level class probabilities
549         logits = outputs[0]
550 
551         # Compute the predicted labels
552         predictions = torch.argmax(logits, dim=-1)
553 
554         # Remove padding and special tokens
555         input_ids = input_ids[0].tolist()
556         predictions = predictions[0].tolist()
557 
558         # real_input_ids = [id for id, pred in zip(
    input_ids, predictions) if id != 0 and id != 101 and
     id != 102]
559         real_predictions = [pred for id, pred in zip
    (input_ids, predictions) if id != 0 and id != 101
    and id != 102]
560 
561         # real_input_ids = []
562         # real_predictions = []
563         #
564         # for id, pred in zip(input_ids, predictions
    ):
565         #     if id != 0 and id != 101 and id != 102
    : # Ignore [PAD], [CLS] and [SEP] tokens
566         #         real_input_ids.append(id)
567         #         real_predictions.append(pred)
568 
569         # Map predicted label indices back to label
    strings
570         predicted_labels = [inv_label_map[label] for
     label in real_predictions]
571 
572         # Combine tokens and their predicted labels
    into dict
573         results = dict(zip(sentence.split(' '),
    predicted_labels))
574 
575         # Put results into final_dict, key is word
    and value is what it is labeled as
576         for word, label in results.items():
577             print(word, label)
```

```
578             if word in ['for', 'and', 'from']:
579                 continue
580             if label != 'O':
581                 # Adds word to proper dictionary if
    belongs to that type
582 
583                 # Checks if there is a B-word before
     the I-word so it will add to it
584 
585                 # It will make a new word if there
    is no B-word
586 
587                 if label == 'B-Product':
588                     final_dict['Product Type'].
    append(word)
589 
590                 elif label == 'I-Product':
591 
592                   if final_dict['Product Type']:
593                     final_dict['Product Type'][-1
    ] += ' ' + word
594                   else:
595                     final_dict['Product Type'].
    append(word)
596 
597                 elif label == 'B-Material':
598                     final_dict['Material Type'].
    append(word)
599 
600                 elif label == 'I-Material':
601 
602                   if final_dict['Material Type']:
603                     final_dict['Material Type'][-1
    ] += ' ' + word
604                   else:
605                      final_dict['Material Type'].
    append(word)
606 
607                 elif label == 'B-Country':
608                     final_dict['Countries'].append(
    word)
```

```
609
610                 elif label == 'I-Country':
611
612                   if final_dict['Countries']:
613                     final_dict['Countries'][-1] +=
    ' ' + word
614                   else:
615                     final_dict['Countries'].append(
    word)
616
617                 elif label == 'B-Application':
618                     final_dict['Building
    applications'].append(word)
619
620                 elif label == 'I-Application':
621
622                   if final_dict['Building
    applications']:
623                     final_dict['Building
    applications'][-1] += ' ' + word
624                   else:
625                     final_dict['Building
    applications'].append(word)
626
627                 elif label == 'B-Recycle':
628                     final_dict['Recycled Content'].
    append(word)
629
630                 elif label == 'I-Recycle':
631
632                     if final_dict['Recycled Content'
    ]:
633                         final_dict['Recycled Content
    '][-1] += ' ' + word
634                     else:
635                         final_dict['Recycled Content
    '].append(word)
636         print('\n')
637
638
639     return final_dict
```

```
#%% md
# Testing
#%% md
## Model Initialization
#%%
Token_Classification_Labels,
    Token_Classification_model =
    Token_Classification_Initialization()
#%%
'''
EXCEL FILE WITH SHEETS OF UNIQUE PRODUCT/MATERIAL/
    BUILDINGAPPLICATION TYPE REDACTED
PROPRIETARY KNOWLEDGE OF 2050 MATERIALS
CONTACT info@2050-materials.com FOR MORE INFORMATION
'''
excel_file = 'YOUR_EXCEL_FILE.xlsx'

product_types, material_types, building_app_list,
    country_list, recycled_list  = list_initialization(
    excel_file)
#%%
queries, labels = new_sentences(1000, product_types
    , material_types, building_app_list, country_list,
    recycled_list, all_words='n')
#%%
tokenizer = Initialize_tokenizer()
#%%
train_dataloader, valid_dataloader, test_dataloader
     = train_data_info(queries=queries, labels=labels,
    tokenizer=tokenizer)
'''
TOKEN CLASSIFICATION MODEL SAVE PATH LOCATION
IF FIRST TIME TRAINING, OR NO FILE IS PASSED,
IT WILL SAVE 'Token_Class_Model.pth' IN THIS FOLDER
    WHEN COMPLETE WITH TRAINING
'''
model_file = 'YOUR_TOKEN_CLASSIFICATION_MODEL.pth'
#%%
raw_results = Token_Classification_Model_Training(
    Token_Classification_Labels=
    Token_Classification_Labels,
```

```
669
    Token_Classification_model=
    Token_Classification_model,
670
    train_dataloader=train_dataloader,
671
    test_dataloader=test_dataloader,
672
    Model_load_file=model_file,
673
    learning_rate=1e-5,
674
    epoch_loop=5)
675 #%%
676 results_dict = results_dict_creation(raw_results)
677 #%%
678 df = pd.DataFrame.from_dict(results_dict)
679 save_results(df)
680 #%%
681
682 for index in range(1, 5):
683     plot_creation(df.iloc[:, 0], df.iloc[:, index])
684 #%%
685 '''
686 CREATE SENTENCE TO TEST MODEL
687 '''
688 list_of_sentences = ['TEST SENTENCE PASSED INTO
    MODEL TO DETERMINE OUTPUT']
689 final_output = Token_Classification_Results(
    List_of_Sentences=list_of_sentences,
690
    Token_Classification_Labels=
    Token_Classification_Labels,
691
    Token_Classification_model=
    Token_Classification_model,
692
    tokenizer=tokenizer,
693
    Token_Class_File=model_file
694                                              )
```

```
695 #%%
696 final_output
697 #%%
698 
```