INTRODUCTION TO DEEP LEARNING – FINAL PROJECT

PROBLEM STATEMENT

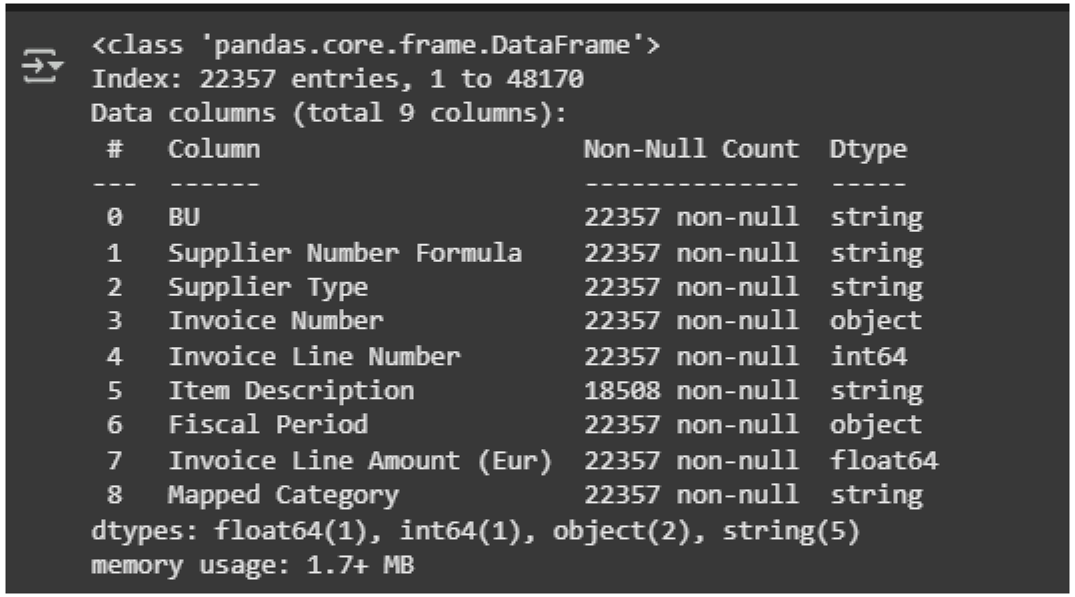
The problem analyzed for the Introduction to Deep Learning final is a text classification problem for transactions at a real-world company. The data can't be shared due to its confidential nature, but all EDA and model training/evaluation and results will be presented.

At this company, transactions mainly get classified in one of two ways; however, it is possible for transactions to process with no classification. In many instances, there are descriptions of what these transactions were, just no official categorization. The resources don't exist to manually review these currently transactions to add classifications after they've initially been processed, and the lack of information causes issues within the company. For this reason, it seemed to be a perfect classification problem to address via machine learning.

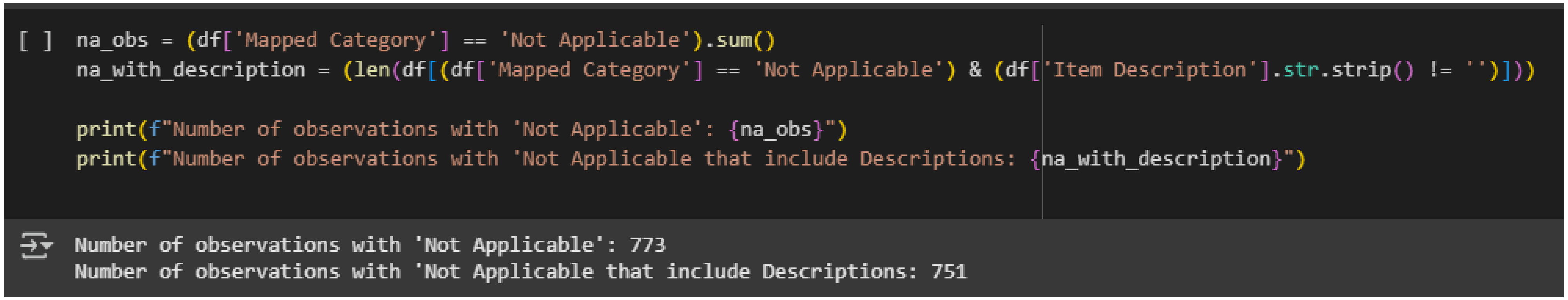
The goal of this project is to use the text from the descriptions of each transaction to help classify those transactions that are not currently classified in our system.

EXPLORATORY DATA ANALYSIS

After the necessary packages and data were imported, EDA was performed. Before proceeding further with this project, the number of uncategorized transactions (those labeled Not Applicable) need to be reviewed to ensure that an adequate number of them include descriptions, the field we'll be using in our text classification model to predict the categories. If they don't, the exercise will be pointless. The below code shows the high-level data frame review as well as the analysis of the Not Applicable transactions. Of 773 uncategorized transactions, 751 of them do include descriptions, which is a large enough sample to continue with this exercise



```
<class 'pandas.core.frame.DataFrame'>
Index: 22357 entries, 1 to 48170
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   BU                        22357 non-null  string
 1   Supplier Number Formula   22357 non-null  string
 2   Supplier Type             22357 non-null  string
 3   Invoice Number            22357 non-null  object
 4   Invoice Line Number       22357 non-null  int64
 5   Item Description          18508 non-null  string
 6   Fiscal Period             22357 non-null  object
 7   Invoice Line Amount (Eur) 22357 non-null  float64
 8   Mapped Category           22357 non-null  string
dtypes: float64(1), int64(1), object(2), string(5)
memory usage: 1.7+ MB
```



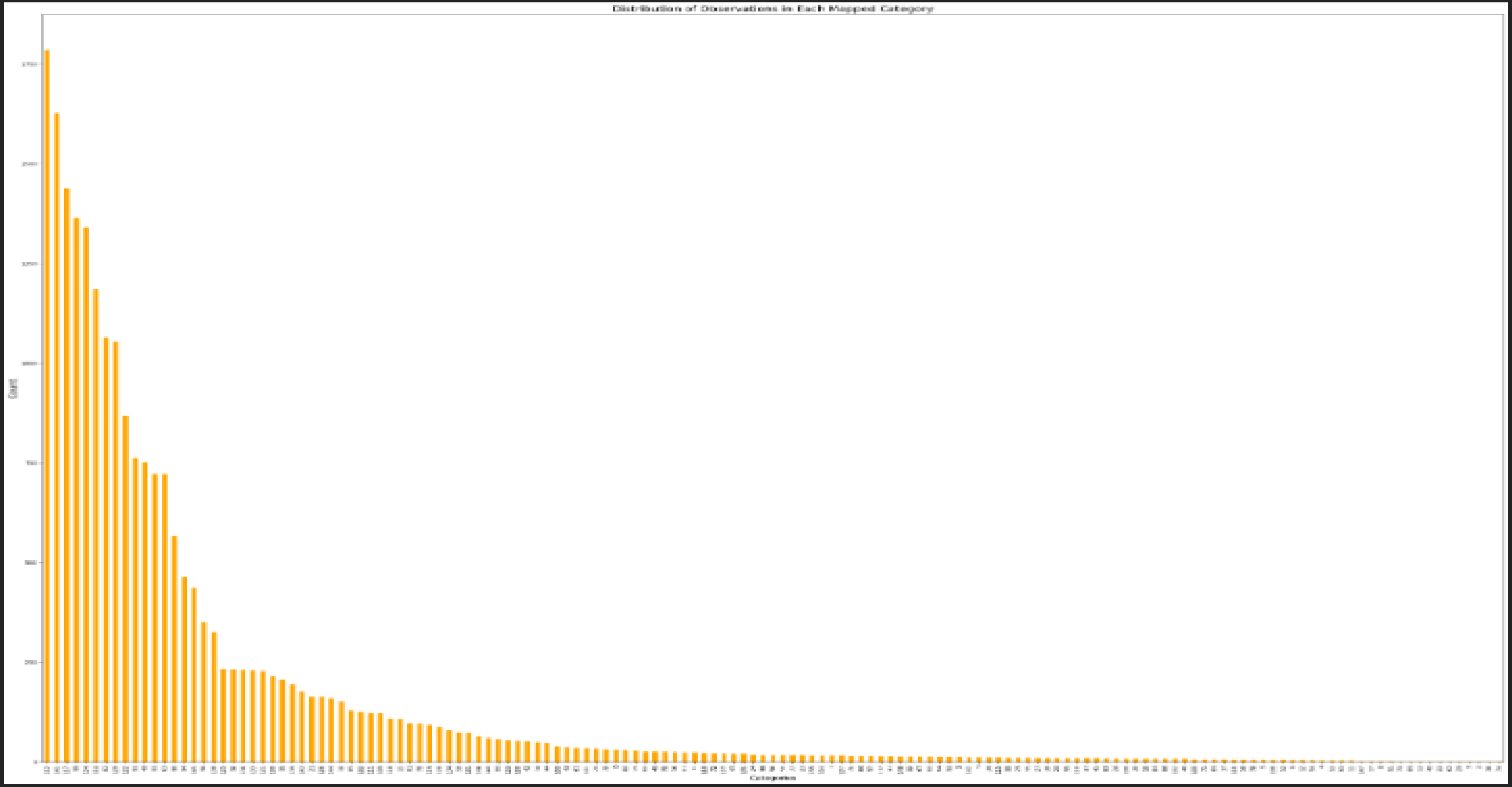
```
na_obs = (df['Mapped Category'] == 'Not Applicable').sum()
na_with_description = (len(df[(df['Mapped Category'] == 'Not Applicable') & (df['Item Description'].str.strip() != '')]))

print(f"Number of observations with 'Not Applicable': {na_obs}")
print(f"Number of observations with 'Not Applicable that include Descriptions: {na_with_description}")
```

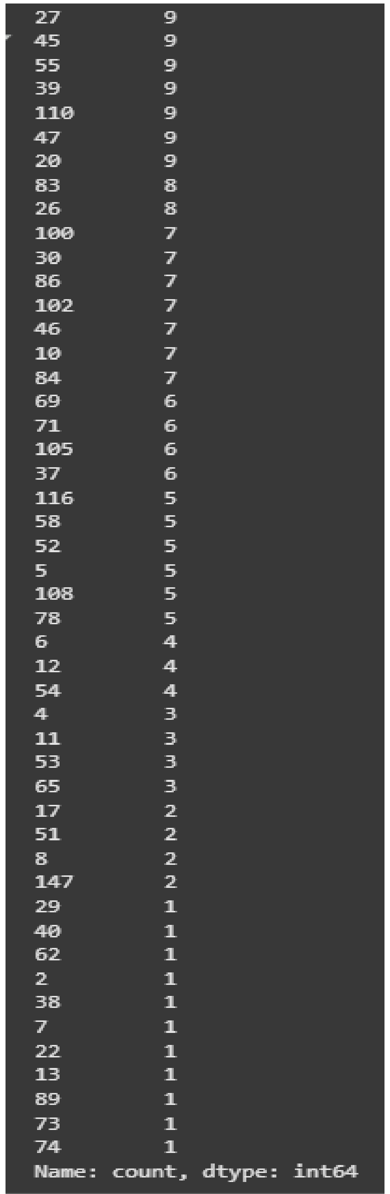
```
Number of observations with 'Not Applicable': 773
Number of observations with 'Not Applicable that include Descriptions: 751
```

The next step in the EDA process was to remove the rows that had neither an assigned category nor an item description value as they won't be useful for prediction. Next the category labels got mapped to numeric representations to keep that data private, then the distribution of transactions per category was analyzed. A total of 179 possible categories exist, 180 if Not Applicable is included. However, the analysis showed that only 149 categories were actually utilized in the data set, and the bar plot shows there is a large discrepancy in the usage of the categories.

```
Number of Mapped Categories actually used = 149
```

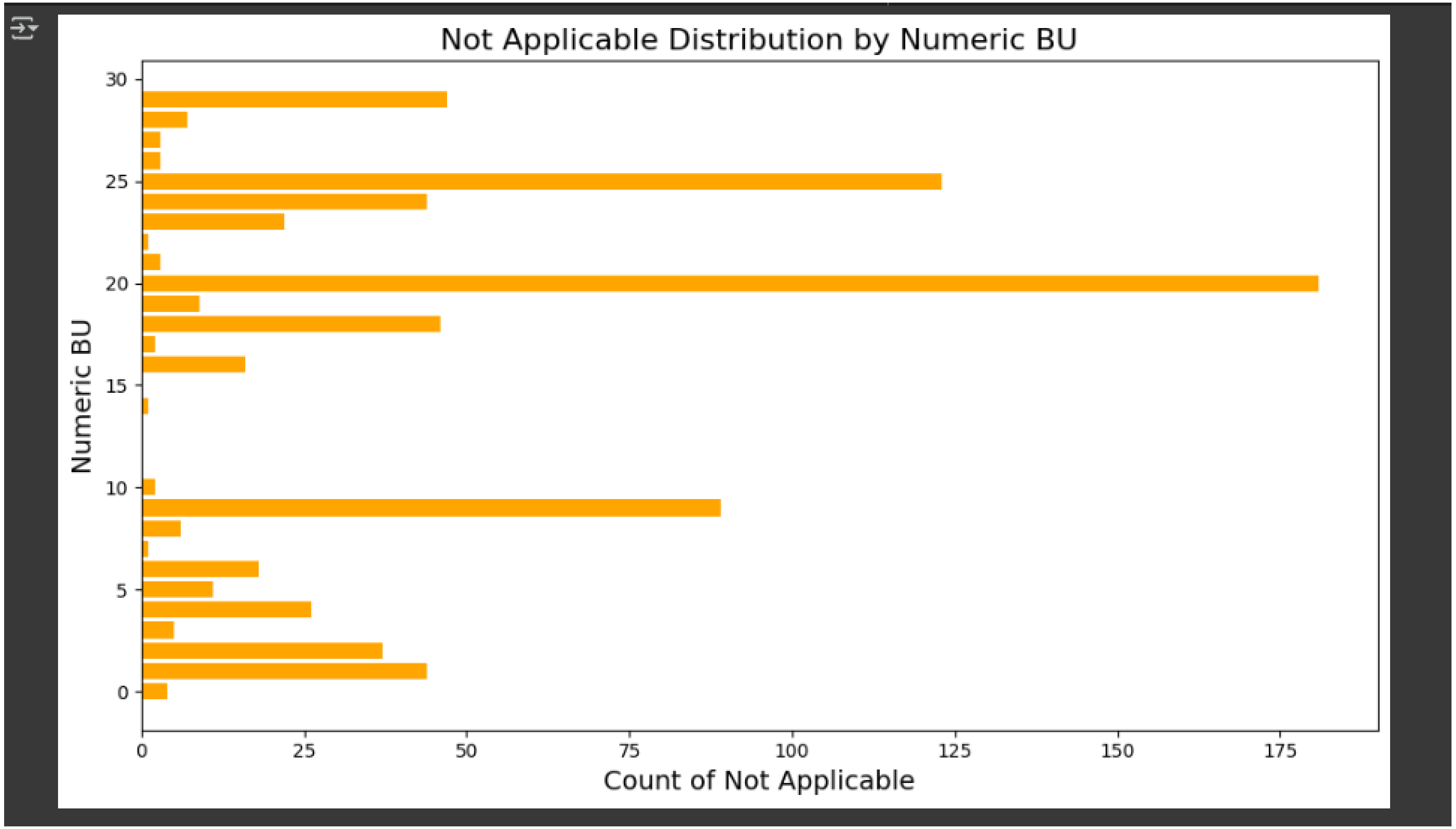


As it’s difficult to discern the labels and values in the bar chart, the actual number of observations per category was printed to the console for further analysis, a truncated view of which is shown below to demonstrate the number of categories with less than 10 observation in our dataset. There are 48 categories out of 149 utilized that were used less than 10 times – 11 categories that have only been utilized once.

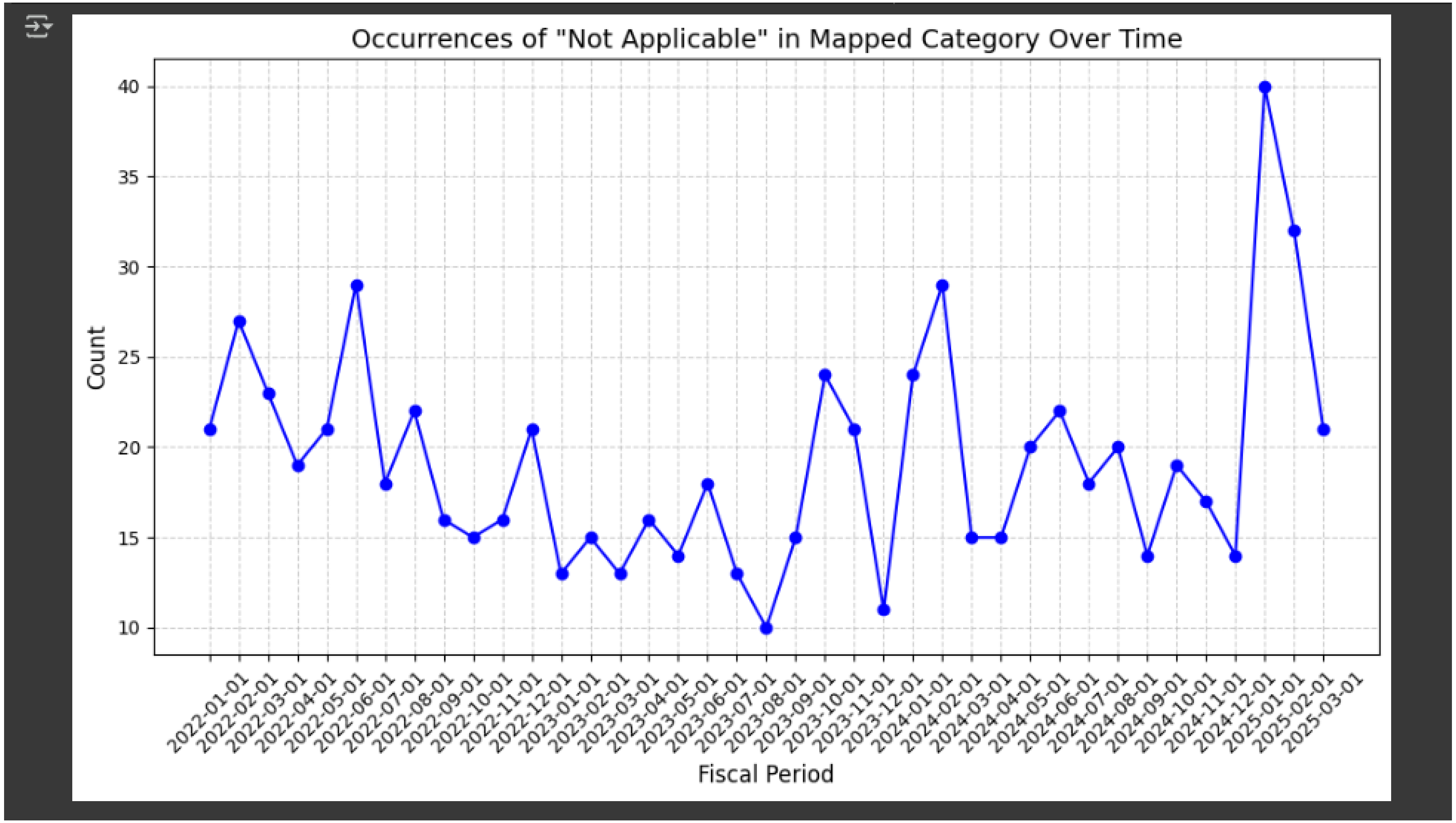


```
27         9
45         9
55         9
39         9
110        9
47         9
20         9
83         8
26         8
100        7
30         7
86         7
102        7
46         7
10         7
84         7
69         6
71         6
105        6
37         6
116        5
58         5
52         5
5          5
108        5
78         5
6          4
12         4
54         4
4          3
11         3
53         3
65         3
17         2
51         2
8          2
147        2
29         1
40         1
62         1
2          1
38         1
7          1
22         1
13         1
89         1
73         1
74         1
Name: count, dtype: int64
```

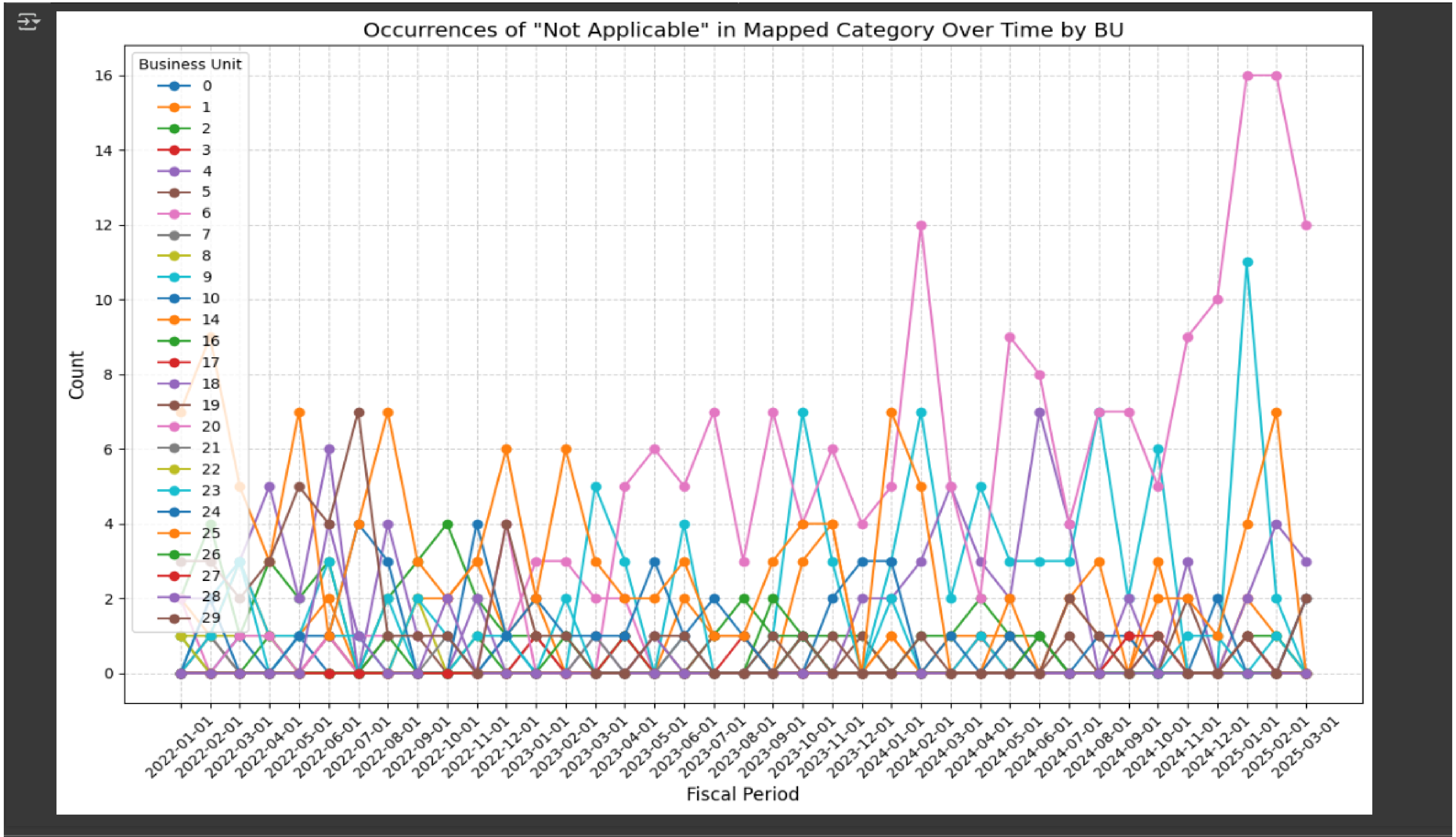
The next step in the EDA was to view a break out of the Not Applicable category usage by region. The plot below shows the breakout by region (the regions have again been mapped to numeric representations for privacy reasons). The large discrepancy in usage indicates there are likely regional issues with the processing of the transactions indicating the issues



A time-series analysis was also performed showing the usage count of the Not Applicable category over time in the data set. Before performing the analysis, the hypothesis was that there would be a downward trend over time as processes and teams matured and learned, and were more apt to correctly classify transactions. That proved not to be the case; there are noticeable spikes around May and January. After investigation, the reason for these spikes was uncovered – the cause is to be kept confidential as well, but this was a very enlightening discovery.



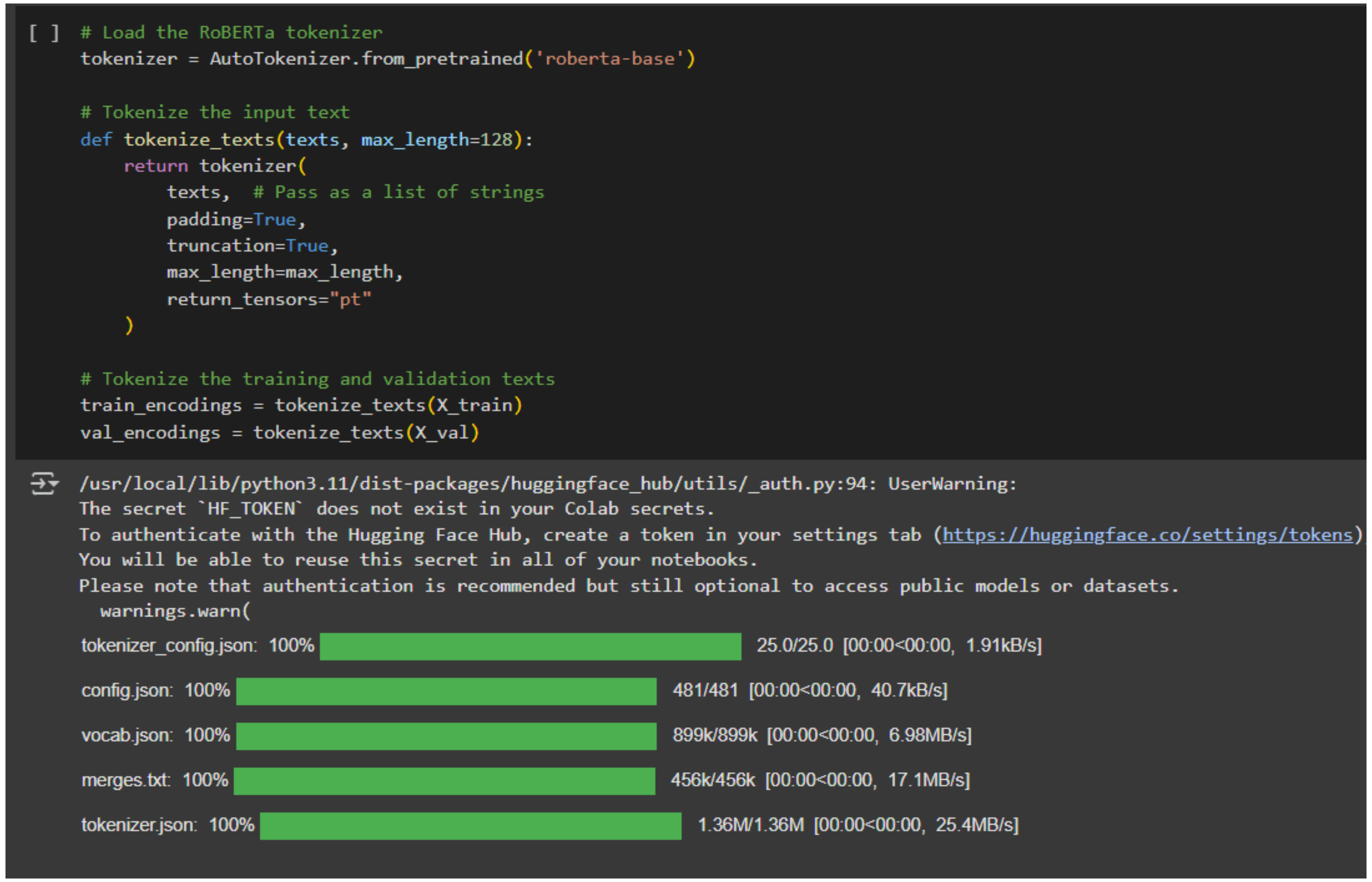
Lastly, the trend over time of Not Applicable usage was broken out by BU to see if any were improving over time. The chart can be a bit difficult to parse through, but there does not appear to be any downward trend for any of the individual BUs. On the contrary, some of the worst offenders appear to have an increasing trend.



ANALYSIS / MODEL TRAINING AND EVALUATION

For this classification task, several models were evaluated, but eventually the RoBERTa model was selected. Like BERT, it is bi-directional and captures the full context of a text sequence, but builds upon and often surpasses BERT in classification problems. It’s more robust and less sensitive to hyperparameter variations as well.

After splitting the provided data into a training and test set (test with no classification) then using sklearn to split the training data further into a training and test data set, the RoBERTa tokenizer was loaded and the data turned into PyTorch data sets.

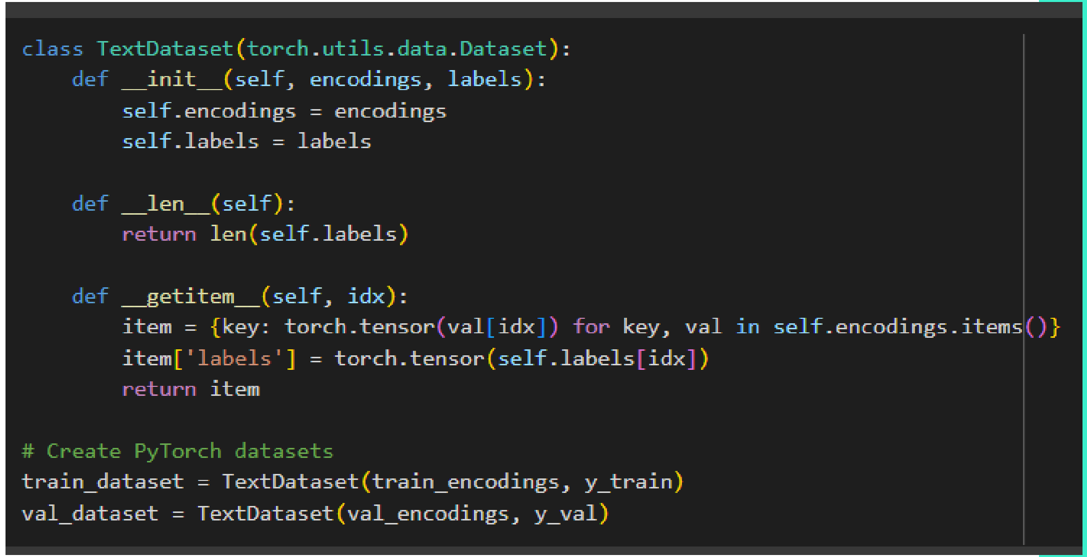


```
[ ] # Load the RoBERTa tokenizer
    tokenizer = AutoTokenizer.from_pretrained('roberta-base')

    # Tokenize the input text
    def tokenize_texts(texts, max_length=128):
        return tokenizer(
            texts,  # Pass as a list of strings
            padding=True,
            truncation=True,
            max_length=max_length,
            return_tensors="pt"
        )

    # Tokenize the training and validation texts
    train_encodings = tokenize_texts(X_train)
    val_encodings = tokenize_texts(X_val)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens)
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100%  25.0/25.0 [00:00<00:00, 1.91kB/s]
config.json: 100%  481/481 [00:00<00:00, 40.7kB/s]
vocab.json: 100%  899k/899k [00:00<00:00, 6.98MB/s]
merges.txt: 100%  456k/456k [00:00<00:00, 17.1MB/s]
tokenizer.json: 100%  1.36M/1.36M [00:00<00:00, 25.4MB/s]
```



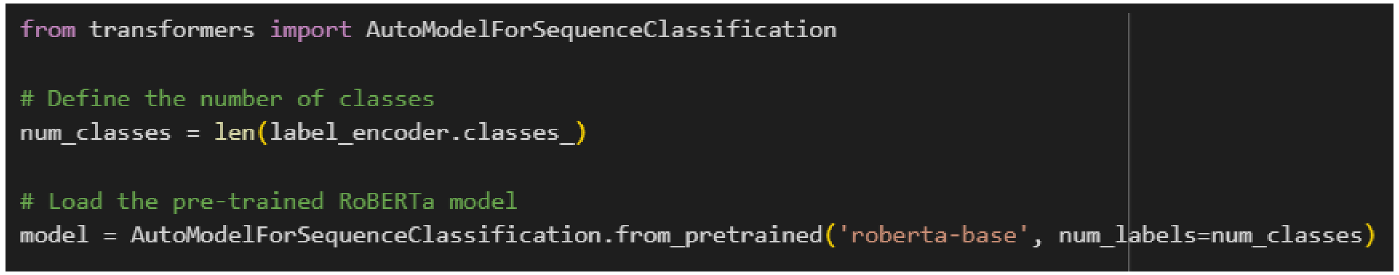
```
class TextDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

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

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

# Create PyTorch datasets
train_dataset = TextDataset(train_encodings, y_train)
val_dataset = TextDataset(val_encodings, y_val)
```

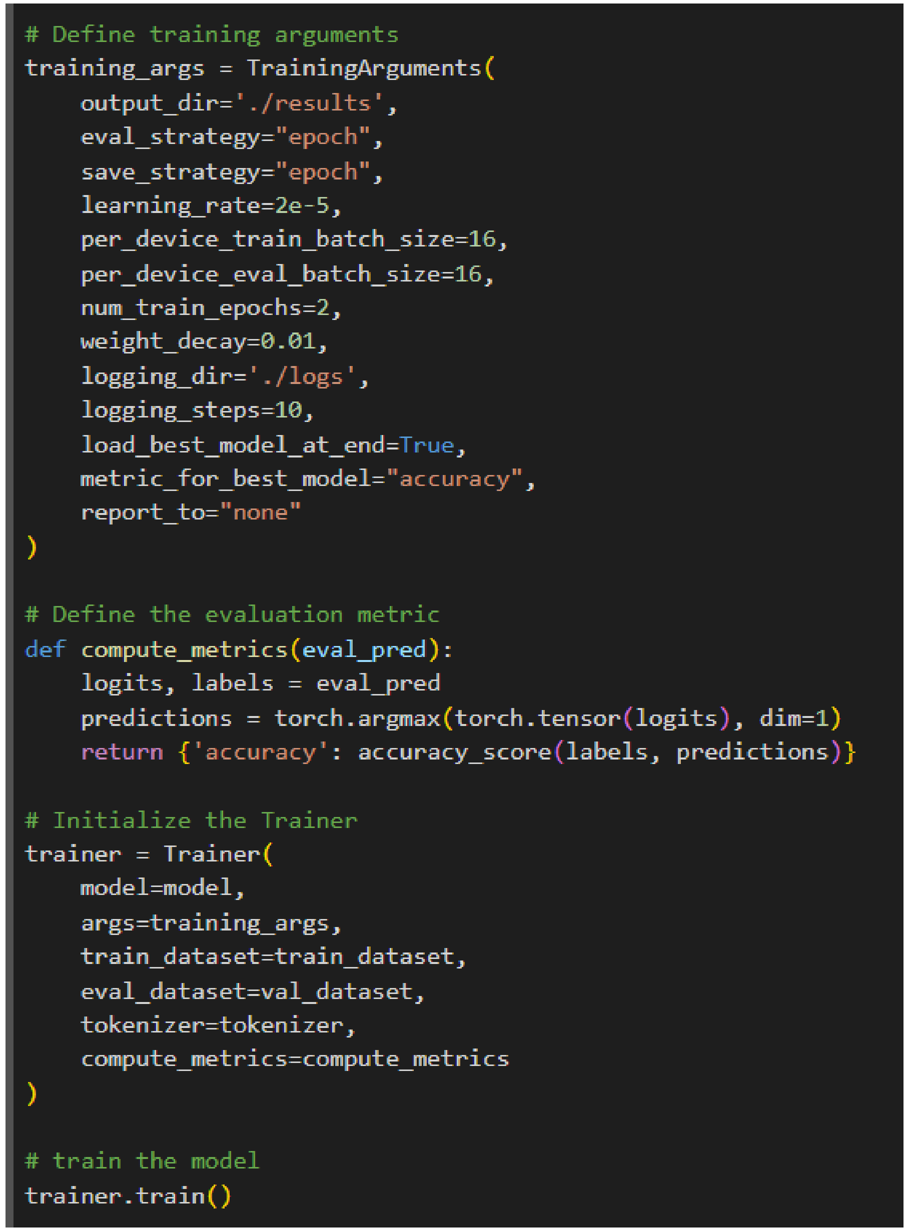
Next, the pre-trained RoBERTa model was loaded, the training arguments and metrics were defined, and the model was trained.



```python
from transformers import AutoModelForSequenceClassification

# Define the number of classes
num_classes = len(label_encoder.classes_)

# Load the pre-trained RoBERTa model
model = AutoModelForSequenceClassification.from_pretrained('roberta-base', num_labels=num_classes)
```



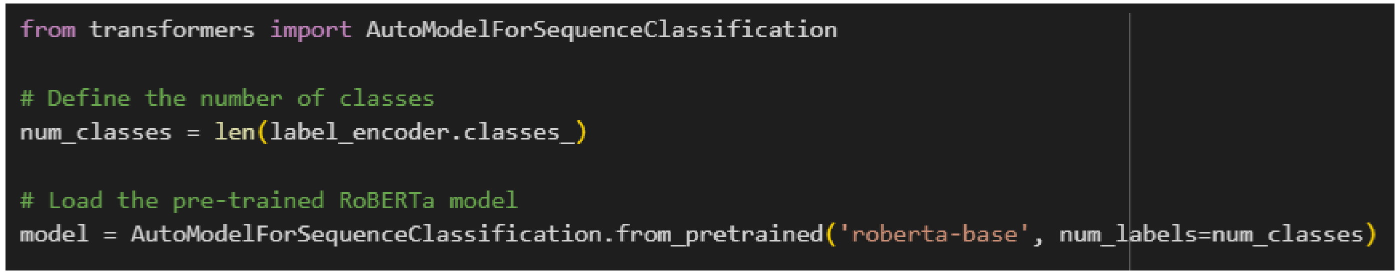
```python
# Define training arguments
training_args = TrainingArguments(
    output_dir='./results',
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=2,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    report_to="none"
)

# Define the evaluation metric
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = torch.argmax(torch.tensor(logits), dim=1)
    return {'accuracy': accuracy_score(labels, predictions)}

# Initialize the Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)

# train the model
trainer.train()
```

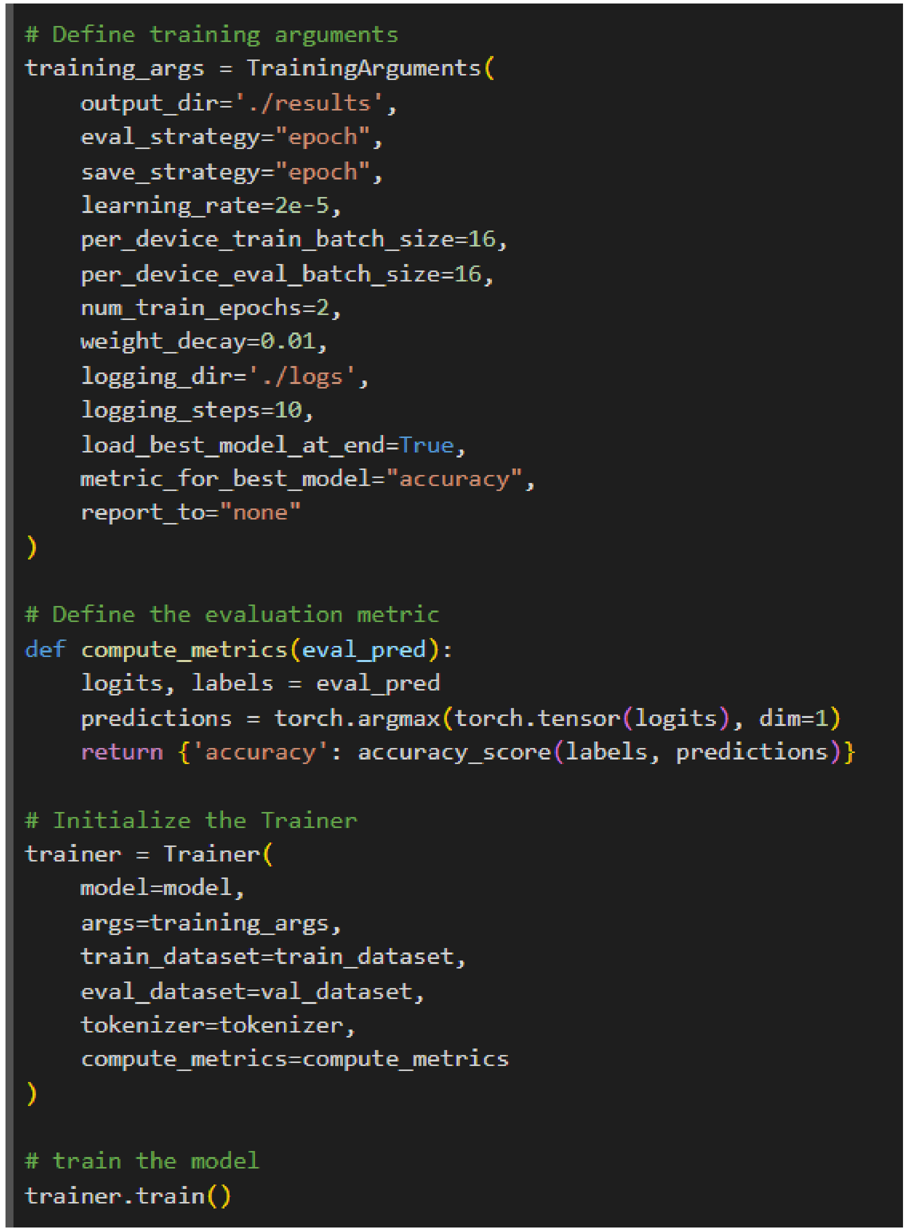
Next, the pre-trained RoBERTa model was loaded, the training arguments and metrics were defined, and the model was trained.



```python
from transformers import AutoModelForSequenceClassification

# Define the number of classes
num_classes = len(label_encoder.classes_)

# Load the pre-trained RoBERTa model
model = AutoModelForSequenceClassification.from_pretrained('roberta-base', num_labels=num_classes)
```

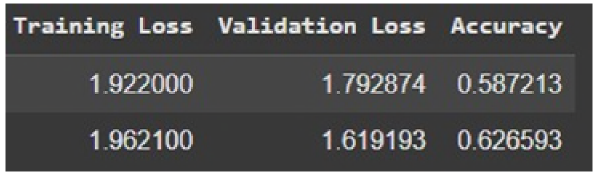


```python
# Define training arguments
training_args = TrainingArguments(
    output_dir='./results',
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=2,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    report_to="none"
)

# Define the evaluation metric
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = torch.argmax(torch.tensor(logits), dim=1)
    return {'accuracy': accuracy_score(labels, predictions)}

# Initialize the Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)

# train the model
trainer.train()
```



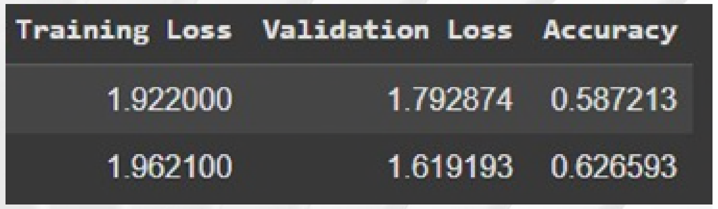
Training Loss	Validation Loss	Accuracy
1.922000	1.792874	0.587213
1.962100	1.619193	0.626593

DISCUSSION & RESULTS

The Training Loss, Validation Loss and Accuracy of the best two epochs are shown in the snip to the right. The model was then used to make predictions on the test data.

To evaluate the accuracy of the model's prediction on the test dataset, SMEs from the company were employed to review the predictions against the categories they would have selected. Note that selecting categories is not an exact science, often there are transactions that could fall under multiple categories, so the SMEs were tasked only with ensuring whether the predicted category was appropriate, not necessarily whether a better option existed. They concluded that ~46% of the predictions were accurate.

Given that the company is currently evaluating a 3rd party solution that's shown accuracy rates in the mid-30% range, this was excellent news! The next step is to harness more powerful resources and work to further improve the models predictions!



Training Loss	Validation Loss	Accuracy
1.922000	1.792874	0.587213
1.962100	1.619193	0.626593

THANK YOU!

LINK TO GITHUB:

https://github.com/mattterry13/MS-DS