In [1]:
```
#!pip install datasets
```

In [5]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

# import datasets
from datasets import list_datasets
all_datasets = list_datasets()
print(f"There are {len(all_datasets)} datasets currently available on the Hub")
print(f"The first 10 are: {all_datasets[:10]}")
```

```
There are 187176 datasets currently available on the Hub
The first 10 are: ['amirveyseh/acronym_identification', 'ade-benchmark-corpus/ade_corpus_v2', 'UCLNLP/adversarial_qa', 'Yale-LILY/aeslc', 'nwu-ctext/afrikaans_ner_corpus', 'fancyzhx/ag_news', 'allenai/ai2_arc', 'google/air_dialogue', 'komari6/ajgt_twitter_ar', 'legacy-datasets/allegro_reviews']
```

In [6]:
```
from datasets import load_dataset
emotions = load_dataset("emotion")
```

In [7]:
```
emotions
```

Out[7]:
```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 16000
    })
    validation: Dataset({
        features: ['text', 'label'],
        num_rows: 2000
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 2000
    })
})
```

In [8]:
```
train_ds = emotions["train"]
train_ds
```

Out[8]:
```
Dataset({
    features: ['text', 'label'],
    num_rows: 16000
})
```

In [9]:
```
len(train_ds)
```

Out[9]:
```
16000
```

In [10]:
```
train_ds[0]
```

Out[10]:
```
{'text': 'i didnt feel humiliated', 'label': 0}
```

In [11]:
```
train_ds.column_names
```

Out[11]:
```
['text', 'label']
```

In [12]:
```
print(train_ds.features)
```

```
{'text': Value(dtype='string', id=None), 'label': ClassLabel(names=['sadness', 'joy', 'love', 'anger', 'fear', 'surprise'], id=None)}
```

In [13]:
```
print(train_ds[:5])
```

```
{'text': ['i didnt feel humiliated', 'i can go from feeling so hopeless to so damned hopeful just from being around someone who cares and is awake', 'im grabbing a minute to post i feel greedy wrong', 'i am ever feeling nostalgic about the fireplace i will know that it is still on the property', 'i am feeling grouchy'], 'label': [0, 0, 3, 2, 3]}
```

In [14]:
```
print(train_ds["text"][:5])
```

```
['i didnt feel humiliated', 'i can go from feeling so hopeless to so damned hopeful just from being around someone who cares and is awake', 'im grabbing a minute to post i feel greedy wrong', 'i am ever feeling nostalgic about the fireplace i will know that it is still on the property', 'i am feeling grouchy']
```

In [15]:
```
import pandas as pd
emotions.set_format(type="pandas")
df = emotions["train"][:]
df.head()
```

Out[15]:

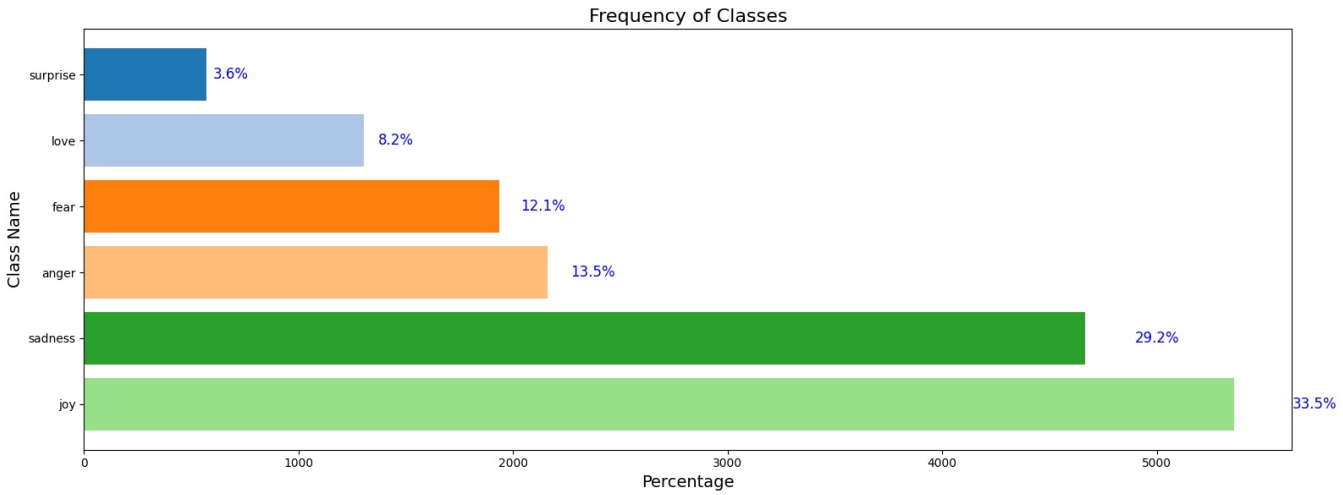
	text	label
0	i didnt feel humiliated	0
1	i can go from feeling so hopeless to so damned...	0
2	im grabbing a minute to post i feel greedy wrong	3
3	i am ever feeling nostalgic about the fireplac...	2
4	i am feeling grouchy	3

In [16]:
```
def label_int2str(row):
    return emotions["train"].features["label"].int2str(row)

df["label_name"] = df["label"].apply(label_int2str)
df.head()
```

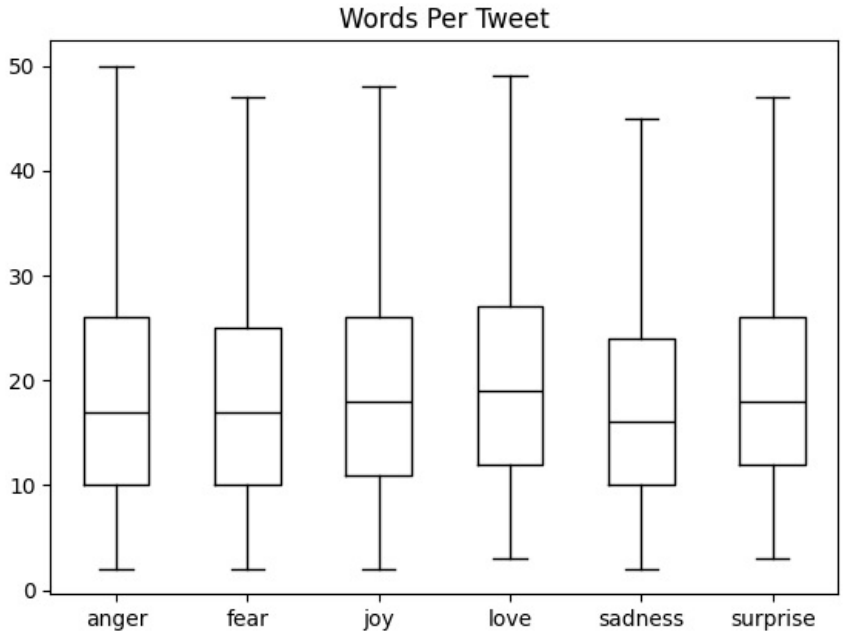
Out[16]:

	text	label	label_name
0	i didnt feel humiliated	0	sadness
1	i can go from feeling so hopeless to so damned...	0	sadness
2	im grabbing a minute to post i feel greedy wrong	3	anger
3	i am ever feeling nostalgic about the fireplac...	2	love
4	i am feeling grouchy	3	anger

In [17]:
```
import matplotlib.pyplot as plt

# Assuming your DataFrame is named 'df'
class_counts = df["label_name"].value_counts(ascending=True)  # Count class frequencies

# Create a list of bright and distinct colors for bars
colors = plt.cm.get_cmap('tab20').colors  # Use a built-in colormap

fig, ax = plt.subplots(figsize=(16, 6))  # Set appropriate figure size

# Create the bar plot with percentage labels
bars = ax.barh(class_counts.index, class_counts.values, color=colors[:len(class_counts)])

# Calculate percentages and format them nicely
percentages = (class_counts / class_counts.sum()) * 100
percentage_labels = [f"{pct:.1f}%" for pct in percentages]

# Increase text size for clarity
plt.xlabel("Percentage", fontsize=14)
plt.ylabel("Class Name", fontsize=14)
for label, rect in zip(percentage_labels, bars):
    ax.text(rect.get_width() + rect.get_width() * 0.05, rect.get_y() + rect.get_height() / 2, label,
            ha='left', va='center', fontsize=12, color='Blue')  # Adjust position and color

plt.title("Frequency of Classes", fontsize=16)
plt.gca().invert_yaxis()  # Arrange bars from top to bottom (optional)
plt.tight_layout()
plt.show()
```



In [18]:
```
df["Words Per Tweet"] = df["text"].str.split().apply(len)
df.boxplot("Words Per Tweet", by="label_name", grid=False,
showfliers=False, color="black")
plt.suptitle("")
plt.xlabel("")
plt.show()
```



In [19]:
```
emotions.reset_format()
```

In [20]:
```
text = "Tokenizing text is a core task of NLP."
tokenized_text = list(text)
print(tokenized_text)
```

```
['T', 'o', 'k', 'e', 'n', 'i', 'z', 'i', 'n', 'g', ' ', 't', 'e', 'x', 't', ' ', 'i', 's', ' ', 'a', ' ', 'c',
'o', 'r', 'e', ' ', 't', 'a', 's', 'k', ' ', 'o', 'f', ' ', 'N', 'L', 'P', '.']
```

In [21]:
```
token2idx = {ch: idx for idx, ch in enumerate(sorted(set(tokenized_text)))}
print(token2idx)
```

```
{' ': 0, '.': 1, 'L': 2, 'N': 3, 'P': 4, 'T': 5, 'a': 6, 'c': 7, 'e': 8, 'f': 9, 'g': 10, 'i': 11, 'k': 12, 'n'
: 13, 'o': 14, 'r': 15, 's': 16, 't': 17, 'x': 18, 'z': 19}
```

In [22]:
```
input_ids = [token2idx[token] for token in tokenized_text]
print(input_ids)
```

```
[5, 14, 12, 8, 13, 11, 19, 11, 13, 10, 0, 17, 8, 18, 17, 0, 11, 16, 0, 6, 0, 7, 14, 15, 8, 0, 17, 6, 16, 12, 0,
14, 9, 0, 3, 2, 4, 1]
```

In [23]:
```
categorical_df = pd.DataFrame(
{"Name": ["Bumblebee", "Optimus Prime", "Megatron"], "Label ID": [0,1,2]})
categorical_df
```

Out[23]:

	Name	Label ID
0	Bumblebee	0
1	Optimus Prime	1
2	Megatron	2

In [24]:
```
pd.get_dummies(categorical_df["Name"])
```

Out[24]:

	Bumblebee	Megatron	Optimus Prime
0	True	False	False
1	False	False	True
2	False	True	False

In [25]:
```
import torch
import torch.nn.functional as F
input_ids = torch.tensor(input_ids)
one_hot_encodings = F.one_hot(input_ids, num_classes=len(token2idx))
one_hot_encodings.shape
```

Out[25]:
```
torch.Size([38, 20])
```

In [26]:
```
print(f"Token: {tokenized_text[0]}")
print(f"Tensor index: {input_ids[0]}")
print(f"One-hot: {one_hot_encodings[0]}")
```

```
Token: T
Tensor index: 5
One-hot: tensor([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [27]:
```
tokenized_text = text.split()
print(tokenized_text)
```

```
['Tokenizing', 'text', 'is', 'a', 'core', 'task', 'of', 'NLP.']
```

In [28]:
```
from transformers import AutoTokenizer
model_ckpt = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
```

In [29]:
```
from transformers import DistilBertTokenizer
distilbert_tokenizer = DistilBertTokenizer.from_pretrained(model_ckpt)
```

In [30]:
```
encoded_text = tokenizer(text)
print(encoded_text)
```

```
{'input_ids': [101, 19204, 6026, 3793, 2003, 1037, 4563, 4708, 1997, 17953, 2361, 1012, 102], 'attention_mask':
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

In [31]:
```
tokens = tokenizer.convert_ids_to_tokens(encoded_text.input_ids)
print(tokens)
```

```
['[CLS]', 'token', '##izing', 'text', 'is', 'a', 'core', 'task', 'of', 'nl', '##p', '.', '[SEP]']
```

In [32]:
```
print(tokenizer.convert_tokens_to_string(tokens))
```

```
[CLS] tokenizing text is a core task of nlp. [SEP]
```

In [33]:
```
tokenizer.vocab_size
```

Out[33]:
```
30522
```

In [34]:
```
tokenizer.model_max_length
```

Out[34]:
```
512
```

In [35]:
```
tokenizer.model_input_names
```

Out[35]:
```
['input_ids', 'attention_mask']
```

In [36]:
```
def tokenize(batch):
    return tokenizer(batch["text"], padding=True, truncation=True)
```

In [37]:
```
print(tokenize(emotions["train"][:2]))
```

```
{'input_ids': [[101, 1045, 2134, 2102, 2514, 26608, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101,
1045, 2064, 2175, 2013, 3110, 2061, 20625, 2000, 2061, 9636, 17772, 2074, 2013, 2108, 2105, 2619, 2040, 14977,
1998, 2003, 8300, 102]], 'attention_mask': [[1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]}
```

In [38]:
```
emotions_encoded = emotions.map(tokenize, batched=True, batch_size=None)
```

In [39]:
```
print(emotions_encoded["train"].column_names)
```

```
['text', 'label', 'input_ids', 'attention_mask']
```

In [40]:
```
from transformers import AutoModel
model_ckpt = "distilbert-base-uncased"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = AutoModel.from_pretrained(model_ckpt).to(device)

import warnings
warnings.filterwarnings('ignore')
```

In [41]:
```
from transformers import TFAutoModel
tf_model = TFAutoModel.from_pretrained(model_ckpt)
```

```
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1722592958.131021    1984 service.cc:145] XLA service 0x591c7744f8f0 initialized for platform TPU (
this does not guarantee that XLA will be used). Devices:
I0000 00:00:1722592958.131076    1984 service.cc:153]   StreamExecutor device (0): TPU, 2a886c8
I0000 00:00:1722592958.131080    1984 service.cc:153]   StreamExecutor device (1): TPU, 2a886c8
I0000 00:00:1722592958.131083    1984 service.cc:153]   StreamExecutor device (2): TPU, 2a886c8
I0000 00:00:1722592958.131086    1984 service.cc:153]   StreamExecutor device (3): TPU, 2a886c8
I0000 00:00:1722592958.131092    1984 service.cc:153]   StreamExecutor device (4): TPU, 2a886c8
I0000 00:00:1722592958.131095    1984 service.cc:153]   StreamExecutor device (5): TPU, 2a886c8
I0000 00:00:1722592958.131098    1984 service.cc:153]   StreamExecutor device (6): TPU, 2a886c8
I0000 00:00:1722592958.131100    1984 service.cc:153]   StreamExecutor device (7): TPU, 2a886c8
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFDistilBertModel: ['vocab_t
ransform.weight', 'vocab_layer_norm.bias', 'vocab_transform.bias', 'vocab_projector.bias', 'vocab_layer_norm.we
ight']
- This IS expected if you are initializing TFDistilBertModel from a PyTorch model trained on another task or wi
th another architecture (e.g. initializing a TFBertForSequenceClassification model from a BertForPreTraining mo
del).
- This IS NOT expected if you are initializing TFDistilBertModel from a PyTorch model that you expect to be exa
ctly identical (e.g. initializing a TFBertForSequenceClassification model from a BertForSequenceClassification
model).
All the weights of TFDistilBertModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFDistilBer
tModel for predictions without further training.
```

In [42]:
```
tf_xlmr = TFAutoModel.from_pretrained("xlm-roberta-base")
```

```
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFXLMRobertaModel: ['lm_head
.dense.bias', 'lm_head.bias', 'lm_head.layer_norm.bias', 'lm_head.dense.weight', 'lm_head.layer_norm.weight']
- This IS expected if you are initializing TFXLMRobertaModel from a PyTorch model trained on another task or wi
th another architecture (e.g. initializing a TFBertForSequenceClassification model from a BertForPreTraining mo
del).
- This IS NOT expected if you are initializing TFXLMRobertaModel from a PyTorch model that you expect to be exa
ctly identical (e.g. initializing a TFBertForSequenceClassification model from a BertForSequenceClassification
model).
All the weights of TFXLMRobertaModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFXLMRobert
aModel for predictions without further training.
```

In [43]:
```
tf_xlmr = TFAutoModel.from_pretrained("xlm-roberta-base", from_pt=True)
```

```
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFXLMRobertaModel: ['lm_head
.dense.bias', 'lm_head.bias', 'lm_head.layer_norm.bias', 'lm_head.dense.weight', 'lm_head.decoder.weight', 'lm_
head.layer_norm.weight']
- This IS expected if you are initializing TFXLMRobertaModel from a PyTorch model trained on another task or wi
th another architecture (e.g. initializing a TFBertForSequenceClassification model from a BertForPreTraining mo
del).
- This IS NOT expected if you are initializing TFXLMRobertaModel from a PyTorch model that you expect to be exa
ctly identical (e.g. initializing a TFBertForSequenceClassification model from a BertForSequenceClassification
model).
All the weights of TFXLMRobertaModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFXLMRobert
aModel for predictions without further training.
```

In [44]:
```
text = "this is a test"
inputs = tokenizer(text, return_tensors="pt")
print(f"Input tensor shape: {inputs['input_ids'].size()}")
```

```
Input tensor shape: torch.Size([1, 6])
```

In [45]:
```
inputs = {k:v.to(device) for k,v in inputs.items()}
with torch.no_grad():
    outputs = model(**inputs)
print(outputs)
```

```
BaseModelOutput(last_hidden_state=tensor([[[-0.1565, -0.1862,  0.0528,  ..., -0.1188,  0.0662,  0.5470],
         [-0.3575, -0.6484, -0.0618,  ..., -0.3040,  0.3508,  0.5221],
         [-0.2772, -0.4459,  0.1818,  ..., -0.0948, -0.0076,  0.9958],
         [-0.2841, -0.3917,  0.3753,  ..., -0.2151, -0.1173,  1.0526],
         [ 0.2661, -0.5094, -0.3180,  ..., -0.4203,  0.0144, -0.2149],
         [ 0.9441,  0.0112, -0.4714,  ...,  0.1439, -0.7288, -0.1619]]]), hidden_states=None, attentions=None)
```

In [46]:
```
outputs.last_hidden_state.size()
```

Out[46]:
```
torch.Size([1, 6, 768])
```

In [47]:
```
outputs.last_hidden_state[:,0].size()
```

Out[47]:
```
torch.Size([1, 768])
```

In [48]:
```
def extract_hidden_states(batch):
    """
    Extracts the last hidden state for the CLS token from a PyTorch model batch.

    Args:
        batch (dict): A dictionary containing model inputs.

    Returns:
        dict: A dictionary with the key "hidden_state" containing a NumPy array
              representing the last hidden state for the CLS token.
    """

    # Place model inputs on the GPU (if available)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    inputs = {k: v.to(device) for k, v in batch.items() if k in tokenizer.model_input_names}

    # Extract last hidden states with disabled gradient calculation
    with torch.no_grad():
        last_hidden_state = model(**inputs).last_hidden_state

    # Return hidden state for CLS token (consider potential modifications)
    return {"hidden_state": last_hidden_state[:, 0].cpu().numpy()}
```

In [49]:
```
emotions_encoded.set_format("torch",
columns=["input_ids", "attention_mask", "label"])
```

In [50]:
```
emotions_hidden = emotions_encoded.map(extract_hidden_states, batched=True)
```

In [51]:
```
emotions_hidden["train"].column_names
```

Out[51]:
```
['text', 'label', 'input_ids', 'attention_mask', 'hidden_state']
```

In [52]:
```
import numpy as np
X_train = np.array(emotions_hidden["train"]["hidden_state"])
X_valid = np.array(emotions_hidden["validation"]["hidden_state"])
y_train = np.array(emotions_hidden["train"]["label"])
y_valid = np.array(emotions_hidden["validation"]["label"])
X_train.shape, X_valid.shape
```

Out[52]:
```
((16000, 768), (2000, 768))
```

In [53]:
```
pip uninstall umap
```

```
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling
parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
WARNING: Skipping umap as it is not installed.
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the sys
tem package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
Note: you may need to restart the kernel to use updated packages.
```

In [54]:
```
pip install umap-learn
```

```
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling
parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
Requirement already satisfied: umap-learn in /usr/local/lib/python3.10/site-packages (0.5.6)
Requirement already satisfied: pynndescent>=0.5 in /usr/local/lib/python3.10/site-packages (from umap-learn) (0
.5.13)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/site-packages (from umap-learn) (1.26.4
)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.10/site-packages (from umap-learn)
(1.5.1)
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.10/site-packages (from umap-learn) (1.14.
0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/site-packages (from umap-learn) (4.66.4)
Requirement already satisfied: numba>=0.51.2 in /usr/local/lib/python3.10/site-packages (from umap-learn) (0.60
.0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.10/site-packages (from numb
a>=0.51.2->umap-learn) (0.43.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/site-packages (from pynndescent>=0.5->
umap-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/site-packages (from scikit-lea
rn>=0.22->umap-learn) (3.5.0)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the sys
tem package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv

[notice] A new release of pip is available: 23.0.1 -> 24.2
[notice] To update, run: pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

In [55]:
```
import umap.umap_ as umap
```

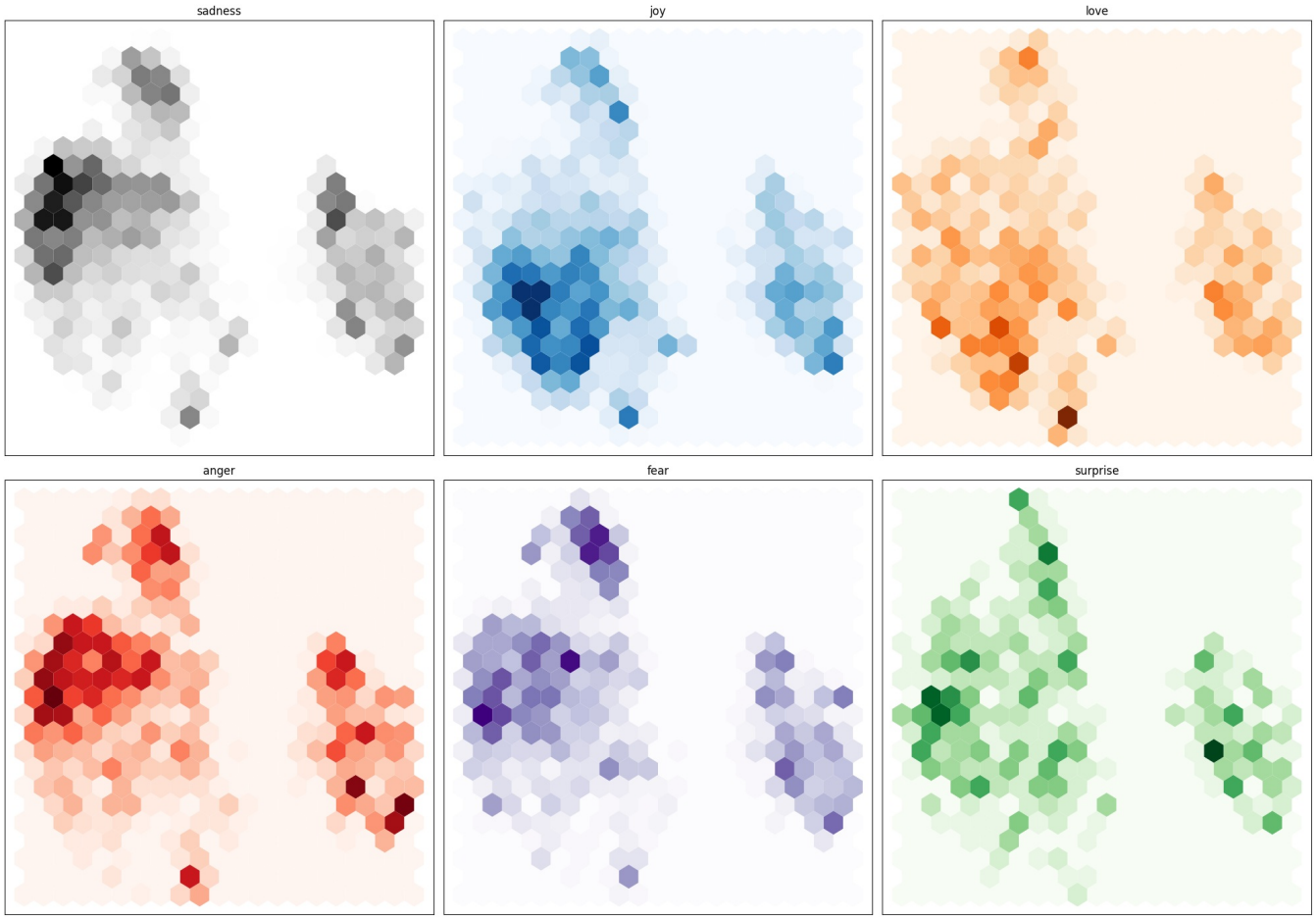
In [56]:
```
from umap import UMAP
from sklearn.preprocessing import MinMaxScaler
# Scale features to [0,1] range
X_scaled = MinMaxScaler().fit_transform(X_train)
# Initialize and fit UMAP
mapper = UMAP(n_components=2, metric="cosine").fit(X_scaled)
# Create a DataFrame of 2D embeddings
df_emb = pd.DataFrame(mapper.embedding_, columns=["X", "Y"])
df_emb["label"] = y_train
df_emb.head()
```

Out[56]:

	X	Y	label
0	4.184766	6.370096	0
1	-3.085471	6.229133	0
2	5.141347	2.963774	3
3	-2.736472	3.799980	2
4	-3.593349	4.304647	3

In [68]:
```
fig, axes = plt.subplots(2, 3, figsize=(20,14))
axes = axes.flatten()
cmaps = ["Greys", "Blues", "Oranges", "Reds", "Purples", "Greens"]
labels = emotions["train"].features["label"].names

for i, (label, cmap) in enumerate(zip(labels, cmaps)):
    df_emb_sub = df_emb.query(f"label == {i}")
```

```
    axes[i].hexbin(df_emb_sub["X"], df_emb_sub["Y"], cmap=cmap,
                   gridsize=20, linewidths=(0,))
    axes[i].set_title(label)
    axes[i].set_xticks([]), axes[i].set_yticks([])

plt.tight_layout()
plt.show()
```



In [58]:
```
from sklearn.linear_model import LogisticRegression
# We increase `max_iter` to guarantee convergence
lr_clf = LogisticRegression(max_iter=3000)
lr_clf.fit(X_train, y_train)
lr_clf.score(X_valid, y_valid)
```

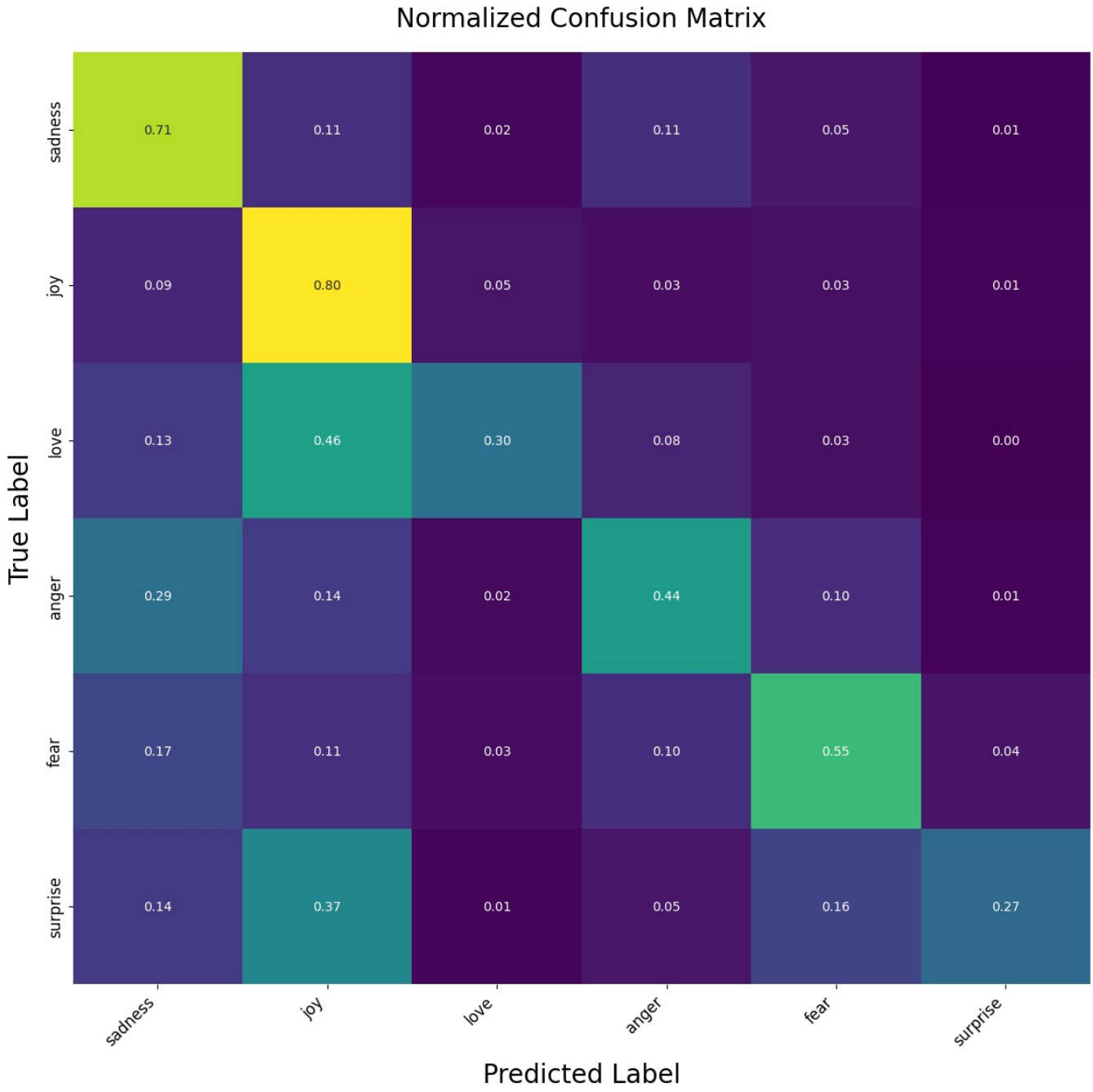
Out[58]:
```
0.6335
```

In [59]:
```
from sklearn.dummy import DummyClassifier
dummy_clf = DummyClassifier(strategy="most_frequent")
dummy_clf.fit(X_train, y_train)
dummy_clf.score(X_valid, y_valid)
```

Out[59]:
```
0.352
```

In [66]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
import seaborn as sns

def plot_confusion_matrix(y_preds, y_true, labels):
    cm = confusion_matrix(y_true, y_preds, normalize="true")

    # Increase figure size
    plt.figure(figsize=(12, 12))

    # Use seaborn to create a heatmap
    sns.heatmap(cm, annot=True, fmt='.2f', cmap='viridis',
                xticklabels=labels, yticklabels=labels, cbar=False)

    plt.title("Normalized Confusion Matrix", fontsize=20, pad=20)
    plt.xlabel('Predicted Label', fontsize=20, labelpad=10)
    plt.ylabel('True Label', fontsize=20, labelpad=10)

    # Increase tick label font size
    plt.xticks(fontsize=12, rotation=45, ha='right')
    plt.yticks(fontsize=12)

    # Adjust layout to prevent cutting off labels
    plt.tight_layout()
```

```
    plt.show()

# Assuming you have y_preds, y_valid, and labels defined
y_preds = lr_clf.predict(X_valid)
plot_confusion_matrix(y_preds, y_valid, labels)
```



https://www.kaggle.com/code/pythonafroz/hugging-face-nlp-02

In [ ]: