**MC1VkYR5W1**

In [1]:
```
import pandas as pd
import numpy as np
import re
y_pred_train = nb_classifier.predict(X_train)
y_pred_val = nb_classifier.predict(X_val)
y_pred_test = nb_classifier.predict(X_test)


from sklearn.decomposition import PCA, TruncatedSVD

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import seaborn as sns
#import cont
```

In [2]:
```

with open('train_text.txt', 'r') as file:
    train_text = file.read().splitlines()

with open('train_labels.txt', 'r') as file:
    train_labels = file.read().splitlines()

with open('val_text.txt', 'r') as file:
    val_text = file.read().splitlines()

with open('val_labels.txt', 'r') as file:
    val_labels = file.read().splitlines()

with open('test_text.txt', 'r') as file:
    test_text = file.read().splitlines()

with open('test_labels.txt', 'r') as file:
    test_labels = file.read().splitlines()

# Now, train_text, train_labels, val_text, val_labels, test_text, and t
# are lists containing the lines from the corresponding files.

# You can then convert them to pandas DataFrames if needed.


train_df = pd.DataFrame({'text': train_text, 'label': train_labels})
val_df= pd.DataFrame({'text': val_text, 'label': val_labels})
test_df= pd.DataFrame({'text': test_text, 'label': test_labels})
```

In [3]:

```python
# Display sample data from each DataFrame
print("Sample data from train_df:")
print(train_df.head())

print("\nSample data from test_df:")
print(test_df.head())

print("\nSample data from val_df:")
print(val_df.head())
```

```
Sample data from train_df:
                                                text  label
0  "QT @user In the original draft of the 7th boo...      2
1  "Ben Smith / Smith (concussion) remains out of...      1
2  Sorry bout the stream last night I crashed out...      1
3  Chase Headley's RBI double in the 8th inning o...      1
4  @user Alciato: Bee will invest 150 million in ...      2

Sample data from test_df:
                                                text  label
0  @user @user what do these '1/2 naked pics' hav...      1
1  OH: “I had a blue penis while I was this” [pla...      1
2  @user @user That's coming, but I think the vic...      1
3  I think I may be finally in with the in crowd ...      2
4  @user Wow,first Hugo Chavez and now Fidel Cast...      0

Sample data from val_df:
                                                text  label
0  Dark Souls 3 April Launch Date Confirmed With ...      1
1  "National hot dog day, national tequila day, t...      2
2  When girls become bandwagon fans of the Packer...      0
3  @user I may or may not have searched it up on ...      1
4  Here's your starting TUESDAY MORNING Line up a...      1
```

In [4]:

```python
# Combine text data from all datasets
all_data = pd.concat([train_df, test_df, val_df])
```

In [5]:

```python
all_data.shape
```

Out[5]:

```
(59899, 2)
```

The data contains 59,899 rows and 2 columns in total

In [6]:
```
all_data
```

Out[6]:

| | text | label |
|---|---|---|
| **0** | "QT @user In the original draft of the 7th boo... | 2 |
| **1** | "Ben Smith / Smith (concussion) remains out of... | 1 |
| **2** | Sorry bout the stream last night I crashed out... | 1 |
| **3** | Chase Headley's RBI double in the 8th inning o... | 1 |
| **4** | @user Alciato: Bee will invest 150 million in ... | 2 |
| **...** | ... | ... |
| **1995** | "LONDON (AP) "" Prince George celebrates his s... | 1 |
| **1996** | Harper's Worst Offense against Refugees may be... | 1 |
| **1997** | Hold on... Sam Smith may do the theme to Spect... | 2 |
| **1998** | Gonna watch Final Destination 5 tonight. I alw... | 1 |
| **1999** | "Interview with Devon Alexander \""""Speed Kil... | 1 |

59899 rows × 2 columns

In [7]:
```
all_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59899 entries, 0 to 1999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    59899 non-null  object
 1   label   59899 non-null  object
dtypes: object(2)
memory usage: 1.4+ MB
```

The data contains 59,899 entries distributed across two columns.The first column is "text" and the second is"label." The "text" column presumably contains textual data, while the "label" contain class for each text entry. Both columns are characterized as having non-null values throughout, suggesting no missing data. The data type for both columns is listed as "object," implying that they contain string values. The memory usage of the DataFrame is reported as approximately 1.4 megabytes.

### EDA

In [8]:
```
all_data.describe()
```

Out[8]:

| | text | label |
|---|---|---|
| **count** | 59899 | 59899 |
| **unique** | 59871 | 3 |
| **top** | Charlie Rose with rich Lowry\u002c Mort Zucker... | 1 |
| **freq** | 3 | 27479 |

In the text column, there are 59,871 unique entries, indicating that some text entries are duplicated. The most frequently occurring text entry, which appears three times, is "Charlie Rose with rich Lowry\u002c Mort Zucker..."

**Lets look at the distribution of labels in the combined data**

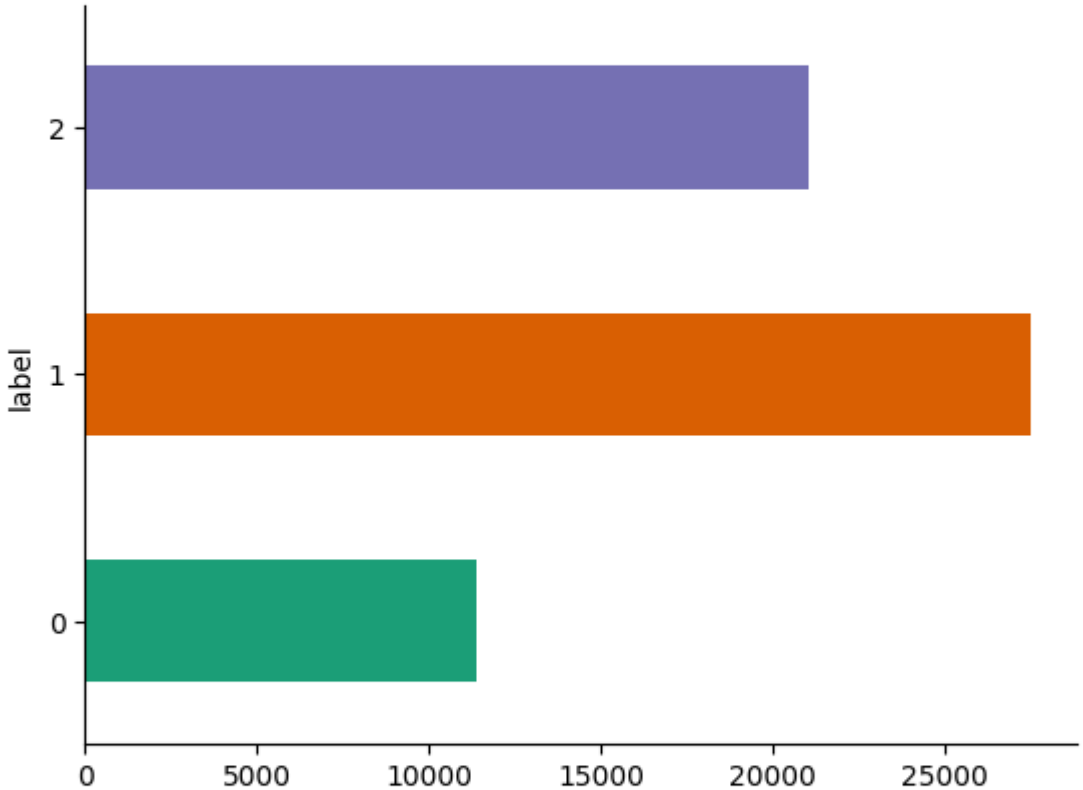
In [9]:
```
# Group the data by 'label' and count the occurrences of 'text' for eac
temp = all_data.groupby('label').count()['text'].reset_index().sort_val
temp.style.background_gradient(cmap='Purples')
```

Out[9]:

| | label | text |
|---|---|---|
| **1** | 1 | 27479 |
| **2** | 2 | 21043 |
| **0** | 0 | 11377 |

We can clearly see that label '1' has the highest occurrence, with 27,479 associated text entries. Label '2' follows with 21,043 text samples, while label '0' has the fewest occurrences, comprising 11,377 text entries.

In [10]:
```
# @title label

from matplotlib import pyplot as plt
import seaborn as sns
all_data.groupby('label').size().plot(kind='barh', color=sns.palettes.m
plt.gca().spines[['top', 'right',]].set_visible(False)
```




**There are more instances of label 1 compared to other labels, indicating that it may be the dominant sentiment category in the data. Label 0 is the least prevalent sentiment category in the dataset. This suggests an imbalance in the distribution of sentiment labels**

**Let's draw a Funnel-Chart for better visualization**

In [11]:
```
import plotly.graph_objects as go

fig = go.Figure(go.Funnelarea(
    text=temp.label,
    values=temp.text,
    title={"position": "top center", "text": "Funnel-Chart of Labels Di
))
fig.show()
```

The funnel chart illustrates a significant class imbalance, with Class 1 having the highest proportion (45.9%), followed by Class 2 (35.1%) and Class 0 (19%). This indicates a skewed distribution of data across the classes, with Class 1 being the most dominant category in the dataset.

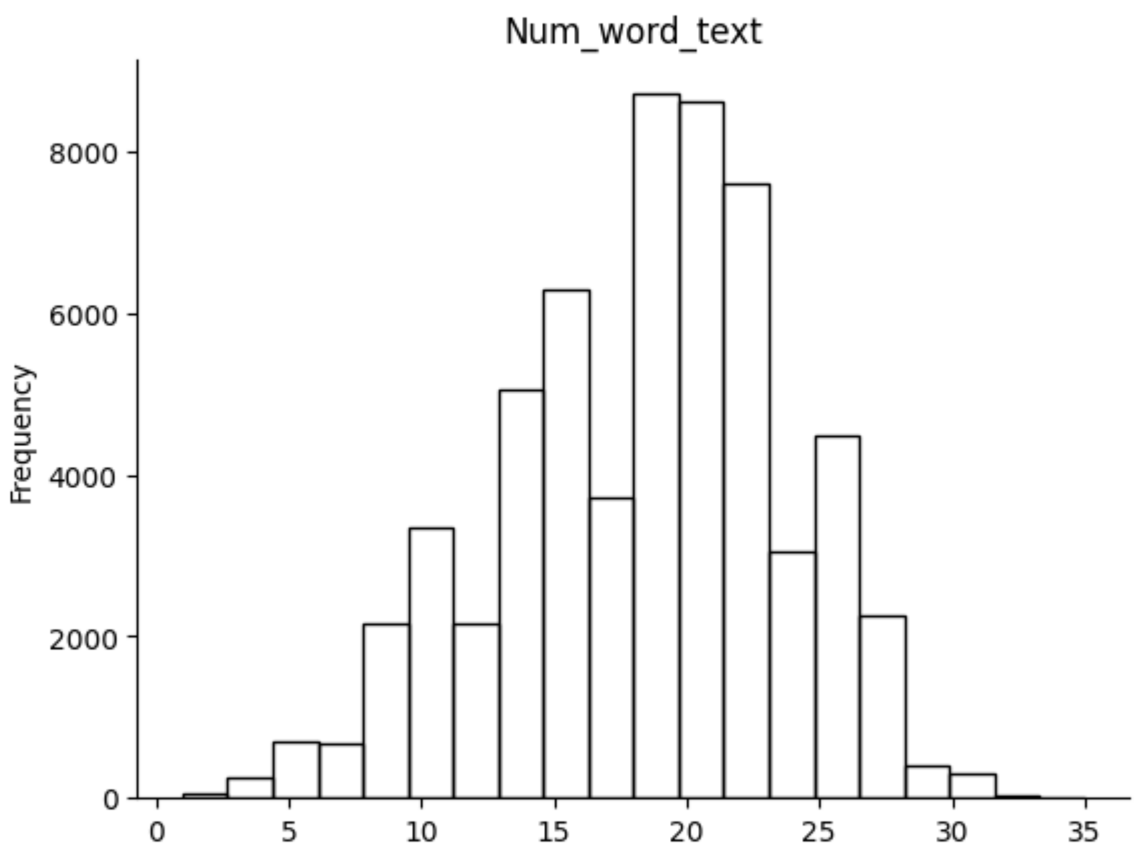
In [12]:
```
#calculates the number of words in the 'text' column of the DataFrame '
all_data['Num_word_text'] = all_data['text'].apply(lambda x:len(str(x).
```

In [13]:
```
all_data.head()
```

Out[13]:

| | text | label | Num_word_text |
|---|---|---|---|
| **0** | "QT @user In the original draft of the 7th boo... | 2 | 18 |
| **1** | "Ben Smith / Smith (concussion) remains out of... | 1 | 14 |
| **2** | Sorry bout the stream last night I crashed out... | 1 | 24 |
| **3** | Chase Headley's RBI double in the 8th inning o... | 1 | 23 |
| **4** | @user Alciato: Bee will invest 150 million in ... | 2 | 21 |

In [14]:
```
# @title Num_word_text

from matplotlib import pyplot as plt

# Plot histogram without fill
all_data['Num_word_text'].plot(kind='hist', bins=20, title='Num_word_te

# Remove top and right spines
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

# Show the plot
plt.show()
```

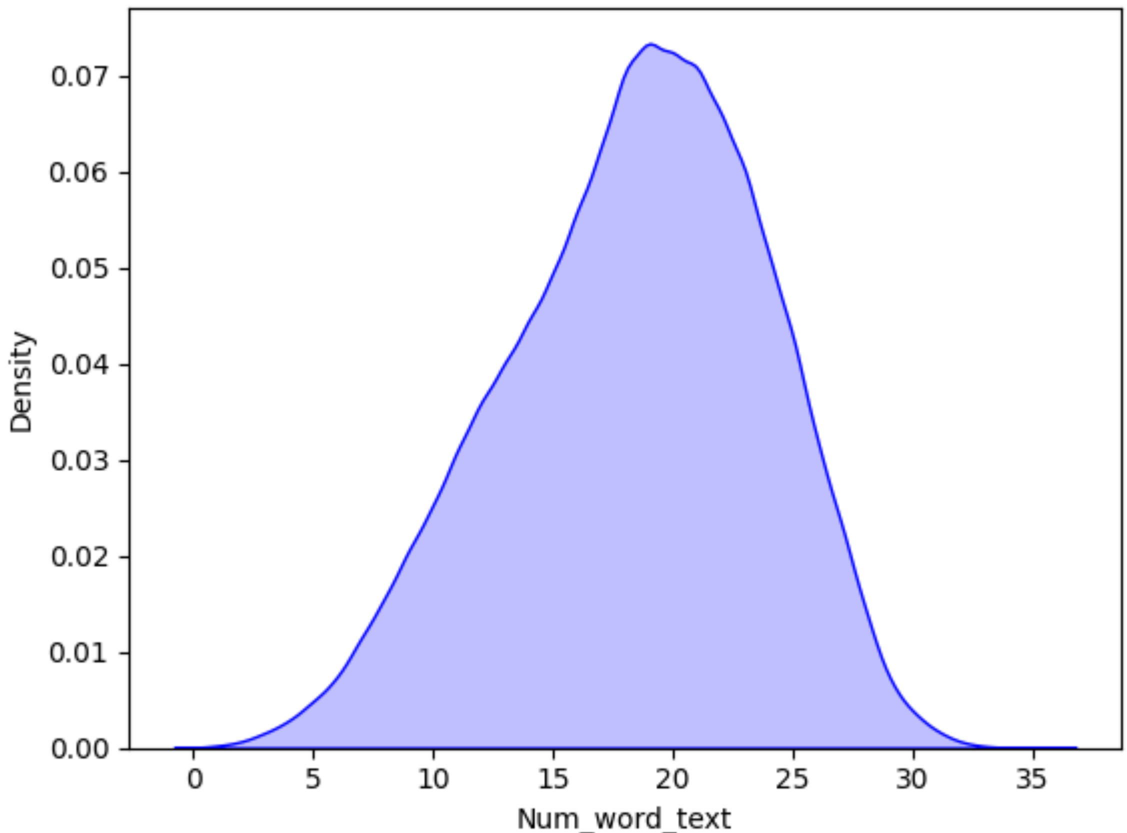



The bar graph displays the distribution of the number of words in the text data. There is a peak around 19 words. A significant portion of the text data is around 15 to 24 words.

In [15]:

```
p1=sns.kdeplot(all_data['Num_word_text'], shade=True, color="b")
```

```
<ipython-input-15-f41e8747f7df>:1: FutureWarning:


`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.
```




The density plot shows that the number of words ranges from 1 to around 35 and they are nomally distributed across

**Most Common words used**

In [16]:
```
from collections import Counter
all_data['temp_list'] = all_data['text'].apply(lambda x:str(x).split())
top = Counter([item for sublist in all_data['temp_list'] for item in su
temp = pd.DataFrame(top.most_common(10))
temp.columns = ['Common_words','count']
temp.style.background_gradient(cmap='Blues')
```

Out[16]:

| | Common_words | count |
|---|---|---|
| **0** | the | 40839 |
| **1** | @user | 25722 |
| **2** | to | 24957 |
| **3** | in | 15491 |
| **4** | a | 15170 |
| **5** | and | 14527 |
| **6** | on | 14088 |
| **7** | of | 13511 |
| **8** | I | 12549 |
| **9** | for | 11712 |

**From the above we can see that the is the most common word used in the text with a count of 40839**

**we didnt remove the stop words and hence we can see the most coomon word is 'the', so we remove the stop words and see**

**Cleaning the Corpus**

In [17]:
```
def find_non_apl_words(text):
    tokens = text.strip().split()
    return [t for t in tokens if not re.match(r'[^\W\d]*$',
                                              t.translate(str.maketrans
                                             )]
```

In [18]:
```
non_alpha_num = []
non_alpha_num = all_data['text'].apply(lambda x: find_non_apl_words(x))
```

In [19]:
```
non_alpha_num
```

Out[19]:
```
0             [7th, #HappyBirthdayRemusLupin"]
1                            [/, #NHL, #SJ"]
2                                         []
3                                  [8th, 33]
4                 [Alciato:, 150, 200, 2017"]
                        ...
1995                                      []
1996                                      []
1997             [#007, #SPECTRE, #JamesBond]
1998                                 [5, :S]
1999                                  [16th]
Name: text, Length: 59899, dtype: object
```

In [20]:
```python
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import string
import re

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

def preprocessing(text):
    # Lowercasing
    text = text.lower()

    # Removing emojis and emoticons
    text = re.sub(r'(:|;|=)(?:-)?(?:\)|\(|D|P)', '', text)
    text = re.sub(r'[\U00010000-\U0010ffff]', '', text)

    # Removing punctuation
    text = ''.join([char for char in text if char not in string.punctua

    # Tokenization
    tokens = nltk.word_tokenize(text)

    # Removing stop words
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]

    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]

    # Removing pronouns
    tokens = [word for word in tokens if word not in ['i', 'you', 'he',

    # Joining tokens back into text
    clean_text = ' '.join(tokens)

    return clean_text
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

In [21]:
```python
all_data['clean_text'] = all_data['text'].apply(preprocessing)
```

In [22]:
```
all_data.head()
```

Out[22]:

| | text | label | Num_word_text | temp_list | clean_text |
|---|---|---|---|---|---|
| **0** | "QT @user In the original draft of the 7th boo... | 2 | 18 | ["QT, @user, In, the, original, draft, of, the... | qt user original draft 7th book remus lupin su... |
| **1** | "Ben Smith / Smith (concussion) remains out of... | 1 | 14 | ["Ben, Smith, /, Smith, (concussion), remains,... | ben smith smith concussion remains lineup thur... |
| **2** | Sorry bout the stream last night I crashed out... | 1 | 24 | [Sorry, bout, the, stream, last, night, I, cra... | sorry bout stream last night crashed tonight s... |
| **3** | Chase Headley's RBI double in the 8th inning o... | 1 | 23 | [Chase, Headley's, RBI, double, in, the, 8th, ... | chase headleys rbi double 8th inning david pri... |
| **4** | @user Alciato: Bee will invest 150 million in ... | 2 | 21 | [@user, Alciato:, Bee, will, invest, 150, mill... | user alciato bee invest 150 million january an... |

In [22]:
```

```

In [23]:
```
from collections import Counter

# Preprocess text data and tokenize it
top = Counter([word for sublist in all_data['clean_text'] for word in s
temp = pd.DataFrame(top.most_common(6))

# Rename columns
temp.columns = ['Common_words', 'count']

# Display DataFrame
temp
```

Out[23]:

| | Common_words | count |
|---|---|---|
| **0** | user | 25760 |
| **1** | tomorrow | 7494 |
| **2** | may | 7002 |
| **3** | day | 4483 |
| **4** | going | 3313 |
| **5** | night | 3236 |

**After removing the stop words, the most common word is user which appears 25760 times followed by tomorrow which appears 7494 times then may which appears 7002 times**

**Visualizing the most common words**

In [24]:
```
import plotly.express as px

fig = px.treemap(temp, path=['Common_words'], values='count', title='Tr
fig.show()
```

In [24]:
```
```

**The treemap above provides a visual summary of the distribution of the most common words in the text and we can clearly see that user appears the most followed by tomorrow**

In [25]:
```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
```

In [26]:
```python
def plot_wordcloud(text, title):
    """ Generate wordcloud"""

    stopwords = set(STOPWORDS)
    comment_words = " ".join(text)  # Join all texts into one string

    wordcloud = WordCloud(
        width = 600,
        height = 600,
        background_color ='white',
        stopwords = stopwords,
        min_font_size = 10
    ).generate(comment_words)

    # plot the WordCloud image
    plt.figure(figsize = (8, 8), facecolor = None)
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.tight_layout(pad = 0)
    plt.title(title, fontsize=20)
    plt.show()
```

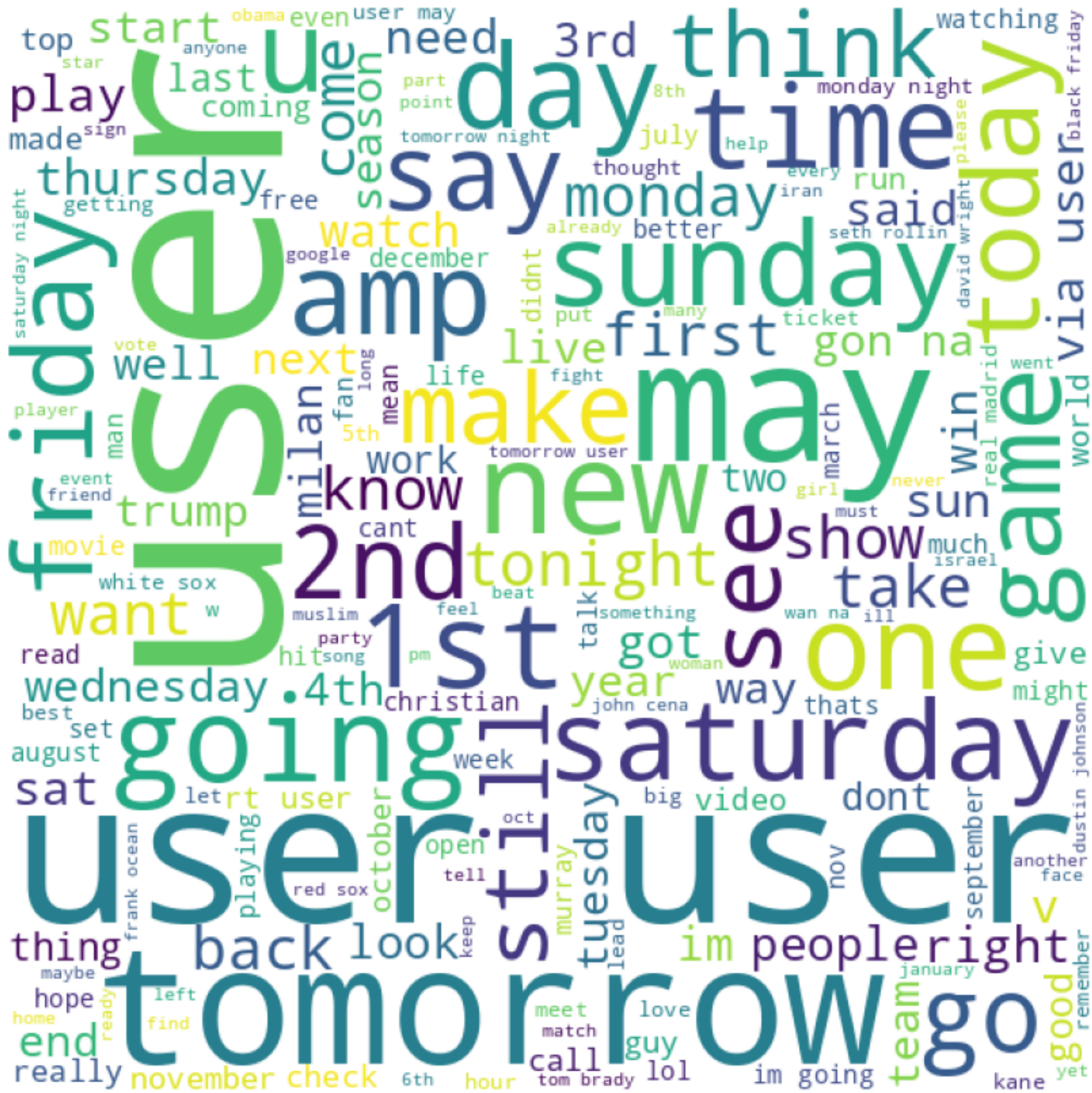
In [27]:

```
# Positive words
title = "Word cloud for label2"
positives = all_data[all_data.label == '2']['clean_text']
plot_wordcloud(positives, title)
```

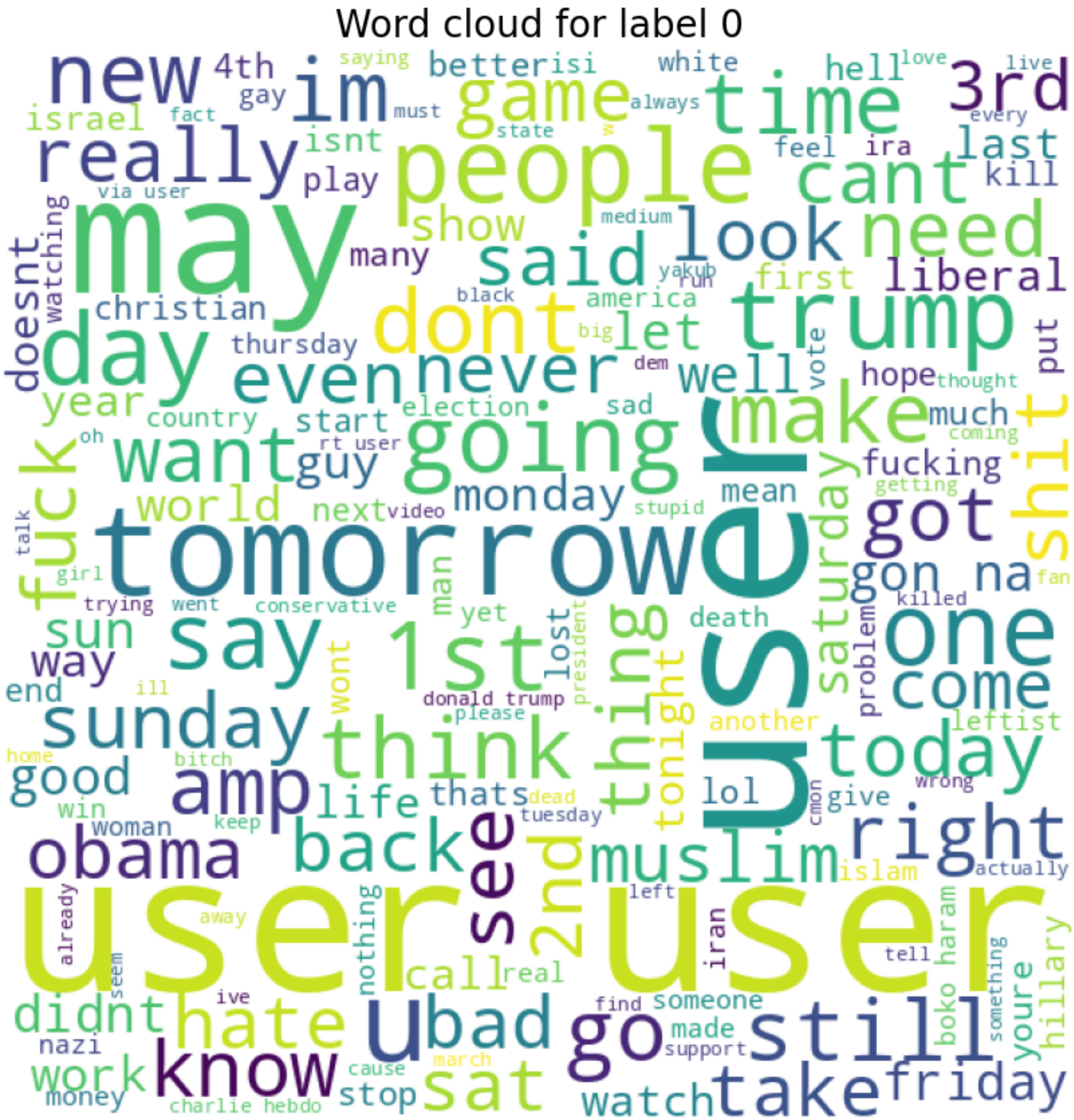
In [28]:

```
# Positive words
title = "Word cloud for label 1"
positives = all_data[all_data.label == '1']['clean_text']
plot_wordcloud(positives, title)
```

Word cloud for label 1



```
In [29]: # Positive words
title = "Word cloud for label 0"
positives = all_data[all_data.label == '0']['clean_text']
plot_wordcloud(positives, title)
```



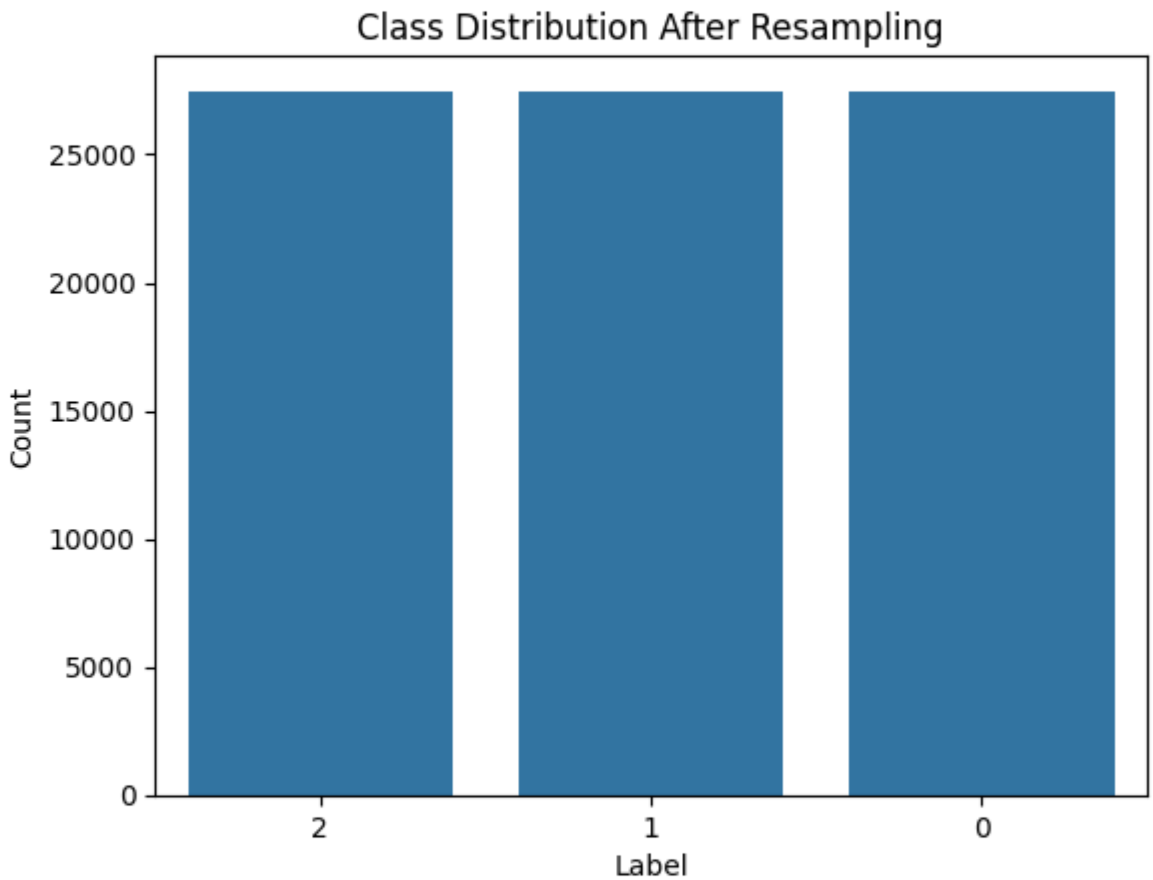
**RESAMPLING**

In [30]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from imblearn.over_sampling import SMOTE

# Create a TF-IDF vectorizer
vectorizer = TfidfVectorizer()

# Fit-transform the text data to convert it into numerical features
X = vectorizer.fit_transform(all_data['clean_text'])

y = all_data['label']

# Create a SMOTE object
smote = SMOTE()

# Resample the dataset
X_resampled, y_resampled = smote.fit_resample(X, y)
```

In [31]:
```
y_resampled.value_counts()
```

Out[31]:
```
2    27479
1    27479
0    27479
Name: label, dtype: int64
```

In [32]:
```
resampled_counts = y_resampled.value_counts().reset_index()
resampled_counts.columns = ['label', 'count']

# Plot the count of each class
ax = sns.barplot(x='label', y='count', data=resampled_counts)
ax.set_title('Class Distribution After Resampling')
ax.set_xlabel('Label')
ax.set_ylabel('Count')
plt.show()
```




After resampling the data is now the data is balanced

In [32]:
```
```

**TRAINING**

In [33]:
```
from tensorflow.keras.preprocessing.text import one_hot
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, LSTM, Dropout, Dense, F
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.utils import plot_model
```

In [34]:
```
X = all_data['clean_text']
y = all_data['label']
```

In [35]:
```
from sklearn.model_selection import train_test_split

# Split data into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Split train set into train and validation sets (60% train, 20% valida
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, tes

# Now you can print the shapes of your data to verify
print("Training data shape:", X_train.shape)
print("Validation data shape:", X_val.shape)
print("Testing data shape:", X_test.shape)
```

```
Training data shape: (35939,)
Validation data shape: (11980,)
Testing data shape: (11980,)
```

The training dataset contains 35,939 samples.

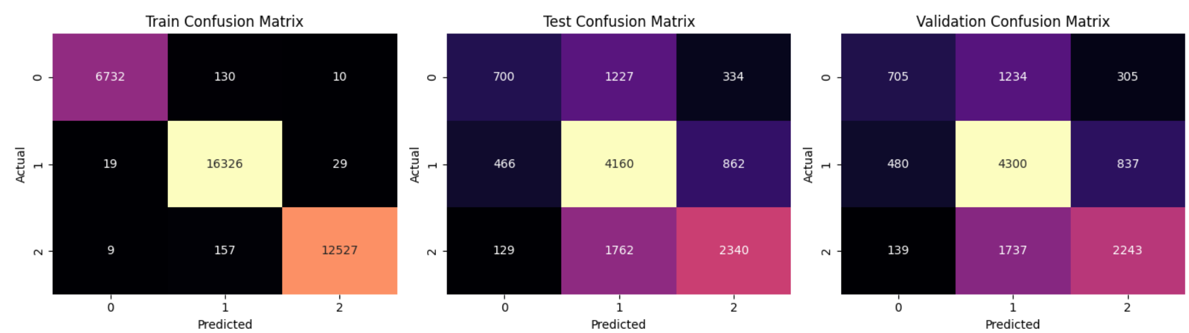
The validation dataset contains 11,980 samples.

The testing dataset contains 11,980 samples.

#### Random Forest Model

In [36]:
```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, ac

# Split data into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Split train set into train and validation sets (60% train, 20% valida
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, tes

# Initialize the TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=1000)  # Adjust max_fea

# Fit and transform the training, testing, and validation text data
X_train_text = tfidf_vectorizer.fit_transform(X_train)
X_test_text = tfidf_vectorizer.transform(X_test)
X_val_text = tfidf_vectorizer.transform(X_val)

# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=4

# Train the classifier
rf_classifier.fit(X_train_text, y_train)

# Predict on training, testing, and validation sets
y_train_pred = rf_classifier.predict(X_train_text)
y_test_pred = rf_classifier.predict(X_test_text)
y_val_pred = rf_classifier.predict(X_val_text)
```

In [37]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Define function to plot confusion matrix
def plot_confusion_matrix(ax, y_true, y_pred, title, cmap='BuPu'):
    cm = confusion_matrix(y_true, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='magma'

, cbar=False, ax=ax)
    ax.set_title(title)
    ax.set_xlabel('Predicted')
    ax.set_ylabel('Actual')

# Create a figure and axes for subplots
fig, axes = plt.subplots(1, 3, figsize=(14, 4))

# Plot confusion matrices for training, testing, and validation sets
plot_confusion_matrix(axes[0], y_train, y_train_pred, 'Train Confusion
plot_confusion_matrix(axes[1], y_test, y_test_pred, 'Test Confusion Mat
plot_confusion_matrix(axes[2], y_val, y_val_pred, 'Validation Confusion

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




**generally the model performs relatively well in correctly classifying instances of classes 0 and 2 across all datasets (train, test, and validation). However, it appears to struggle more with class 1, particularly in the test and validation datasets, where there are higher counts of both false positives and false negatives. Train Confusion Matrix:**

**False Positives (FP) and False Negatives (FN) in Confusion Matrices are:**

Train:

False Positives (FP): 130, 10, 157
False Negatives (FN): 19, 29, 9

Test:

False Positives (FP): 1227, 334, 1762
False Negatives (FN): 466, 129, 837

Validation:

False Positives (FP): 1234, 305, 139

In [38]:
```
# Print classification report for testing set
print("Classification report for testing set:")
print(classification_report(y_test, y_test_pred))
```

```
Classification report for testing set:
              precision    recall  f1-score   support

           0       0.54      0.31      0.39      2261
           1       0.58      0.76      0.66      5488
           2       0.66      0.55      0.60      4231

    accuracy                           0.60     11980
   macro avg       0.59      0.54      0.55     11980
weighted avg       0.60      0.60      0.59     11980
```

For class 0, the precision is 0.54, indicating that 54% of the instances predicted as class 0 are actually class 0. Similarly, for class 1 and class 2, the precisions are 0.58 and 0.66, respectively.

For class 0, the recall is 0.31, meaning that only 31% of the actual class 0 instances were correctly identified by the classifier. For class 1 and class 2, the recalls are 0.76 and 0.55, respectively.

The F1-score ranges from 0 to 1, with higher values indicating better performance. Class 1 has the highest F1-score (0.66), followed by class 2 (0.60) and class 0 (0.39).

Support: It represents the number of actual occurrences of each class in the testing set.There seems to be some class imbalance, as indicated by the variation in support (the number of instances for each class). Class 1 has the highest support (5488), followed by class 2 (4231), and class 0 (2261).

The overall accuracy for this classification is 0.60, meaning that the classifier correctly predicted the class for 60% of the instances in the testing set.

Macro avg: It calculates the average of the metrics (precision, recall, and F1-score) for all classes without considering class imbalance. The macro avg F1-score is 0.55.

Weighted avg: It calculates the average of the metrics, weighted by the support (the number of true instances for each class). The weighted avg F1-score is 0.59.

In [39]:
```
# Calculate and print accuracy scores for training, testing, and valida
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
val_accuracy = accuracy_score(y_val, y_val_pred)
print("Train accuracy score:", train_accuracy)
print("Test accuracy score:", test_accuracy)
print("Validation accuracy score:", val_accuracy)
```

```
Train accuracy score: 0.9901499763488133
Test accuracy score: 0.6010016694490818
Validation accuracy score: 0.605008347245409
```

The model exhibits exceptional accuracy on the training data, achieving a score of approximately 99%. However, this high performance does not carry over to unseen data, as evidenced by the significantly lower accuracy scores of around 60% on the testing and validation sets. This suggests potential overfitting.The model's ability to predict class labels for new instances is limited, indicating the need for further evaluation and refinement.

In [40]:
```
from sklearn.metrics import roc_auc_score

# Calculate and print ROC-AUC scores for testing and validation sets
test_roc_auc = roc_auc_score(y_test, rf_classifier.predict_proba(X_test
val_roc_auc = roc_auc_score(y_val, rf_classifier.predict_proba(X_val_te
print("Test ROC-AUC score:", test_roc_auc)
print("Validation ROC-AUC score:", val_roc_auc)
```

```
Test ROC-AUC score: 0.756752794832347
Validation ROC-AUC score: 0.7505128547256955
```

**Both the test and validation sets have ROC-AUC scores close to 0.75, which suggests that the model has moderate to good discrimination ability in distinguishing between the positive and negative classes.**

**Naive Bayes**

In [47]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, confusion_matrix, ac
from sklearn.model_selection import train_test_split

# Initialize the TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()

# Fit and transform the training text data
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)

# Initialize and train the Naive Bayes classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_tfidf, y_train)
```

Out[47]:
```
MultinomialNB()
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [49]:
```
# Preprocess the text data (assuming X_train, X_val, X_test are your te
X_train_tfidf = tfidf_vectorizer.transform(X_train)
X_val_tfidf = tfidf_vectorizer.transform(X_val)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Make predictions
y_pred_train = nb_classifier.predict(X_train_tfidf)
y_pred_val = nb_classifier.predict(X_val_tfidf)
y_pred_test = nb_classifier.predict(X_test_tfidf)
```

In [50]:
```

print("Training Set Classification Report:")
print(classification_report(y_train, y_pred_train))
```

```
Training Set Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.22      0.36      6872
           1       0.66      0.92      0.77     16374
           2       0.82      0.74      0.78     12693

    accuracy                           0.72     35939
   macro avg       0.82      0.63      0.64     35939
weighted avg       0.78      0.72      0.69     35939
```

For class 0, the precision is high (0.97), indicating that when the model predicts a sample as class 0, it is correct 97% of the time. For class 1 and class 2, the precisions are 0.66 and 0.82 respectively, suggesting that the model's predictions for these classes are also relatively accurate.

The recall for class 0 is low (0.22), meaning that the model only correctly identifies 22% of the actual class 0 samples. However, for class 1 and class 2, the recall values are higher (0.92 and 0.74 respectively), indicating that the model performs better in correctly capturing these classes.

Class 0 has the lowest F1-score (0.36), while class 1 and class 2 have higher F1-scores (0.77 and 0.78 respectively), reflecting their better balance between precision and recall.

The overall accuracy of the model on the training set is 0.72, indicating that it correctly predicts the class for 72% of the samples.

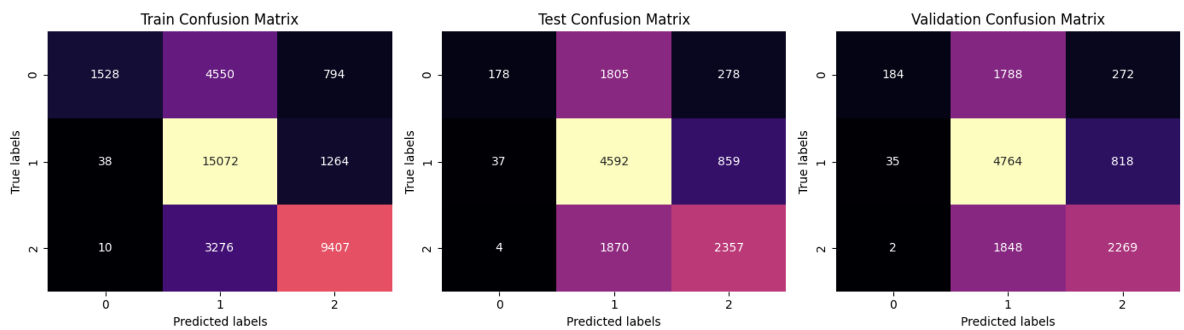
In [51]:

```
# Calculate accuracy for train, test, and validation data
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)
validation_accuracy = accuracy_score(y_val, y_pred_val)

print("Accuracy on Training Set:", train_accuracy)
print("Accuracy on Test Set:", test_accuracy)
print("Accuracy on Validation Set:", validation_accuracy)
```

```
Accuracy on Training Set: 0.723642839255405
Accuracy on Test Set: 0.5949081803005009
Accuracy on Validation Set: 0.6024207011686143
```

With a training set accuracy of 72.36%, the model demonstrates relatively good predictive capability on the data it was trained on. However, the lower accuracy on the test set (59.49%) suggests a decrease in performance when applied to new, unseen data, indicating potential overfitting. Despite this, the model maintains a validation set accuracy of 60.24%, indicating consistency in performance across different unseen datasets.

In [62]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Define function to plot confusion matrix
def plot_confusion_matrix(ax, y_true, y_pred, title):
    sns.heatmap(confusion_matrix(y_true, y_pred), annot=True, fmt="d",
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
    ax.set_title(title)

# Plot confusion matrices side by side
fig, axs = plt.subplots(1, 3, figsize=(14, 4))

plot_confusion_matrix(axs[0], y_train, y_pred_train, 'Train Confusion M
plot_confusion_matrix(axs[1], y_test, y_pred_test, 'Test Confusion Matr
plot_confusion_matrix(axs[2], y_val, y_pred_val, 'Validation Confusion

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




**In the training set:**

Class 0: 1528 instances were correctly classified, while 4550 instances of class 1 were misclassified as class 0, and 794 instances of class 2 were misclassified as class 0.

Class 1: 15072 instances were correctly classified, while 38 instances of class 0 were misclassified as class 1, and 1264 instances of class 2 were misclassified as class 1.

Class 2: 9407 instances were correctly classified, while 10 instances of class 0 were misclassified as class 2, and 3276 instances of class 1 were misclassified as class 2.

**In the test set:**

Class 0: 178 instances were correctly classified, while 1805 instances of class 1 were misclassified as class 0, and 278 instances of class 2 were misclassified as class 0.

Class 1: 4592 instances were correctly classified, while 37 instances of class 0 were misclassified as class 1, and 859 instances of class 2 were misclassified as class 1.

Class 2: 2357 instances were correctly classified, while 4 instances of class 0 were misclassified as class 2, and 1870 instances of class 1 were misclassified as class 2.

**In the validation set:**

Class 0: 184 instances were correctly classified, while 1788 instances of class 1 were misclassified as class 0, and 272 instances of class 2 were misclassified as class 0.

Class 1: 4764 instances were correctly classified, while 35 instances of class 0 were misclassified as class 1, and 818 instances of class 2 were misclassified as class 1.

Class 2: 2269 instances were correctly classified, while 2 instances of class 0 were

In [53]:
```
from sklearn.metrics import roc_auc_score

# Calculate and print ROC-AUC scores for testing and validation sets
test_roc_auc = roc_auc_score(y_test, rf_classifier.predict_proba(X_test
val_roc_auc = roc_auc_score(y_val, rf_classifier.predict_proba(X_val_te
print("Test ROC-AUC score:", test_roc_auc)
print("Validation ROC-AUC score:", val_roc_auc)
```

```
Test ROC-AUC score: 0.756752794832347
Validation ROC-AUC score: 0.7505128547256955
```

A ROC-AUC score of 0.756 for the test set and 0.751 for the validation set indicates that the model performs relatively well in distinguishing between the classes, with higher scores suggesting better discrimination ability. These scores suggest that the model's predictions are above random chance and demonstrate reasonable performance in classification tasks.

In [61]:
```
# Calculate confusion matrices
conf_matrix_train = confusion_matrix(y_train, y_pred_train)
conf_matrix_test = confusion_matrix(y_test, y_pred_test)
conf_matrix_val = confusion_matrix(y_val, y_pred_val)
print("train confution",conf_matrix_train)

print("test confution",conf_matrix_test)
print("val confution",conf_matrix_val)
```

```
train confution [[ 1528  4550   794]
 [   38 15072  1264]
 [   10  3276  9407]]
test confution [[ 178 1805  278]
 [  37 4592  859]
 [   4 1870 2357]]
val confution [[ 184 1788  272]
 [  35 4764  818]
 [   2 1848 2269]]
```

In [56]:

In [ ]: