Student Information

Name: Lin,Pei-Yun

Student ID: 113086852

GitHub ID: u105500014-wq

Instructions

First Phase Submission

1. First: do the **take home** exercises in the DM2025-Lab1-Master that considered as **phase 1 (from exercise 1 to exercise 15)**. You can answer in the master file. **This part is worth 10% of your grade.**

2. Second: follow the same process from the DM2025-Lab1-Master on **the new dataset** up **until phase 1**. You can skip some exercises if you think some steps are not necessary. However main exercises should be completed. You don't need to explain all details as we did (some **minimal comments** explaining your code are useful though). **This part is worth 15% of your grade.**

   - Use the new dataset. The dataset contains a 16 columns including 'text' and 'label', with the sentiment labels being: 1.0 is positive, 0.0 is neutral and -1.0 is negative. You can simplify the dataset and use only the columns that you think are necessary.

   - You are allowed to use and modify the `helper` functions in the folder of the first lab session (notice they may need modification) or create your own.

   - Use this file to complete the homework from the second part. Make sure the code can be run from the beginning till the end and has all the needed output.

3. Third: please attempt the following tasks on **the new dataset**. **This part is worth 10% of your grade.**

   - Generate meaningful **new data visualizations**. Refer to online resources and the Data Mining textbook for inspiration and ideas.

4. Fourth: It's hard for us to follow if your code is messy, so please **tidy up your notebook** and **add minimal comments where needed**. **This part is worth 5% of your grade.**

You can submit your homework following these guidelines: DM2025-Lab1-announcement. Make sure to commit and save your changes to your repository

BEFORE the deadline (September 28th 11:59 pm, Sunday).

Second Phase Submission

You can keep the answer for phase 1 for easier running and update the phase 2 on the same page.

1. First: Continue doing the **take home** exercises in the DM2025-Lab1-Master for **phase 2, starting from Finding frequent patterns**. Use the same master(.ipynb) file. Answer from phase 1 will not be considered at this stage. You can answer in the master file. **This part is worth 10% of your grade.**

2. Second: Continue from first phase and do the same process from the DM2025-Lab1-Master on **the new dataset** for phase 2, starting from Finding frequent pattern. You can skip some exercises if you think some steps are not necessary. However main exercises should be completed. You don't need to explain all details as we did (some **minimal comments** explaining your code are useful though). **This part is worth 15% of your grade.**

   - Continue using this file to complete the homework from the second part. Make sure the code can be run from the beginning till the end and has all the needed output. Use the same new dataset as in phase 1.

   - You are allowed to use and modify the `helper` functions in the folder of the first lab session (notice they may need modification) or create your own.

3. Third: please attempt the following tasks on **the new dataset**. **This part is worth 20% of your grade.**

   - Use this file to answer.
   - Generate **TF-IDF features** from the tokens of each text. This will generating a document matrix, however, the weights will be computed differently (using the TF-IDF value of each word per document as opposed to the word frequency). Refer to this Scikit-learn guide .
   - Implement a simple **Naive Bayes classifier** that automatically classifies the records into their categories. Use both the TF-IDF features and word frequency features to build two seperate classifiers. Note that for the TF-IDF features you might need to use other type of NB classifier different than the one in the Master Notebook. Comment on the differences and when using augmentation with feature pattern. Refer to this article.
4. Fourth: In the lab, we applied each step really quickly just to illustrate how to work with your dataset. There are somethings that are not ideal or the most efficient/meaningful. Each dataset can be handled differently as well. What are those inefficent parts you noticed? How can you improve the Data preprocessing for these specific datasets? **This part is worth 10% of your grade.**

5. Fifth: It's hard for us to follow if your code is messy, so please **tidy up your notebook** and **add minimal comments where needed**. **This part is worth 5%**

of your grade.

You can submit your homework following these guidelines: DM2025-Lab1-announcement. Make sure to commit and save your changes to your repository **BEFORE the deadline (October 19th 11:59 pm, Sunday)**.

Phase 1

In [50]:

```
### Begin Assignment Here
#
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import random

df = pd.read_csv("./newdataset/Reddit-stock-sentiment.csv")
df
```

Out[50]:

	type	datetime	post_id	subreddit	title	auth
0	comment	2025-04-11 17:29:56	mmli62w	wallstreetbets	Retardation is on the menu boys! WSB is so back	StickyTip4
1	comment	2025-04-12 1:12:19	mmnu7v9	wallstreetbets	Retail giant TARGET has now declined for 10 co...	Comfortabl Dog-84
2	comment	2025-04-10 15:09:41	mmeevio	StockMarket	How do you feel about a sitting president maki...	Btankersly
3	post	2023-08-30 17:12:55	165kllm	stockstobuytoday	Who knows more? $VMAR	emiljen
4	comment	2025-04-11 14:48:05	mmkl6bw	StockMarket	The Trump administration is begging Xi Jinping...	Just-Big64
...	...	...	...	...	...	
842	comment	2021-06-30 4:06:06	h3iv6pq	stockstobuytoday	$MRIN Marin Software killed it today. Hope som...	Ordinar Office91
843	comment	2025-04-11 5:01:24	mmijiuz	StockMarket	$ U.S. dollar value (crashing)	lulububu
844	post	2025-03-24 12:30:39	1jipi4v	stockstobuytoday	Analyst Recommendations	saas
845	comment	2025-04-11 20:13:26	mmmely7	wallstreetbets	Weekend Discussion Thread for the Weekend of A...	yes_ur_wro
846	comment	2025-04-12 3:09:06	mmobyz1	wallstreetbets	Someone post the hotline please.	I_am_Nerm

847 rows × 16 columns

In [51]:

```
#
df = df[["text", "sentiment", "label"]].copy()

#
label_map = {1.0: "positive", 0.0: "neutral", -1.0: "negative"}
df["category"] = df["label"]
df["category_name"] = df["label"].map(label_map)

#
df.head()
```

Out[51]:

	text	sentiment	label	category	category_name
0	Calls on retards	-1.0	-1.0	-1.0	negative
1	Stunt as in like why did they even make a big ...	1.0	0.0	0.0	neutral
2	Seeing lots of red in the ticker.	0.0	0.0	0.0	neutral
3	Vision Marine Technologies Inc. is rewriting t...	1.0	1.0	1.0	positive
4	He didn't say thank you.	0.0	-1.0	-1.0	negative

In [52]:

```
#                   847
print("Number of records:", len(df))

#   label
print(df["category_name"].value_counts())

#
print(df.sample(6, random_state=42))
```

```
Number of records: 847
category_name
neutral     423
negative    315
positive    109
Name: count, dtype: int64
                                                  text  sentiment  label
\
457                                          "We" who?        0.0    0.0
342                                     Chicken jockey       -1.0    0.0
280                                      Not great Bob        1.0   -1.0
275  Speak for yourself, my wife is being harder on...        1.0    0.0
843                             Where can I read this?        0.0    0.0
734                       That's a chart since April 6        0.0    0.0

     category category_name
457       0.0       neutral
342       0.0       neutral
280      -1.0      negative
275       0.0       neutral
843       0.0       neutral
734       0.0       neutral
```

In [53]:

```
#
print(df.isnull().sum())

#     text
df = df.dropna(subset=["text"])
```

```
text             0
sentiment        0
label            0
category         0
category_name    0
dtype: int64
```

In [54]:

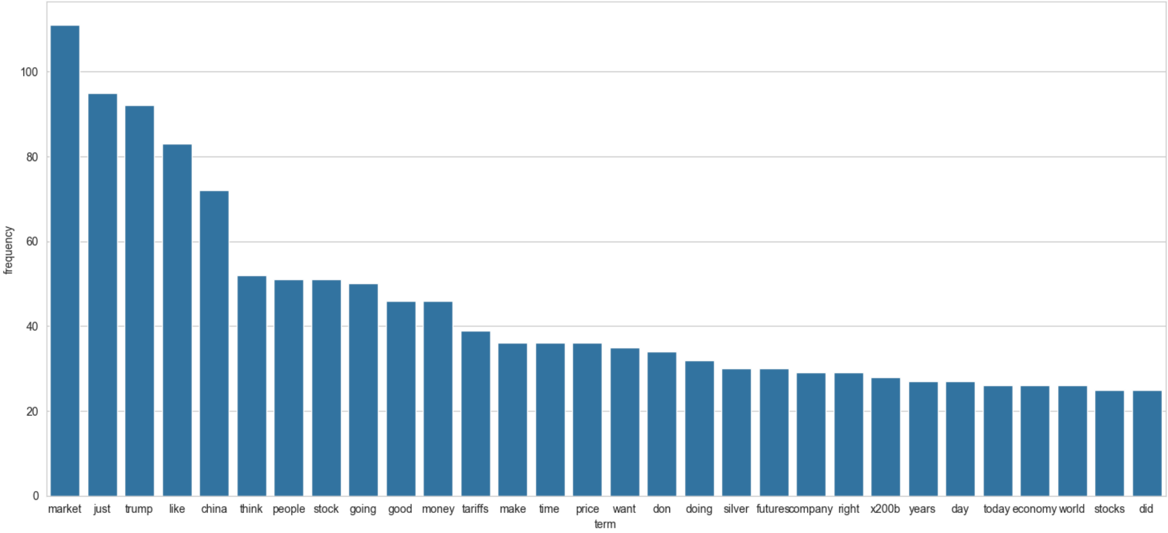
```
#                     ,
from sklearn.feature_extraction.text import CountVectorizer
```

```
count_vect = CountVectorizer(stop_words="english")
X_counts = count_vect.fit_transform(df["text"])

print("Shape of term-document matrix:", X_counts.shape)
```

```
Shape of term-document matrix: (847, 4105)
```

In [55]:

```
#
term_frequencies = np.asarray(X_counts.sum(axis=0))[0]

#
terms = count_vect.get_feature_names_out()
df_terms = pd.DataFrame({"term": terms, "frequency": term_frequencies})
df_terms = df_terms.sort_values(by="frequency", ascending=False)

#  "  30"
plt.figure(figsize=(18,8))
sns.barplot(x="term", y="frequency", data=df_terms.head(30))
plt.xticks(rotation=0)
plt.show()
```



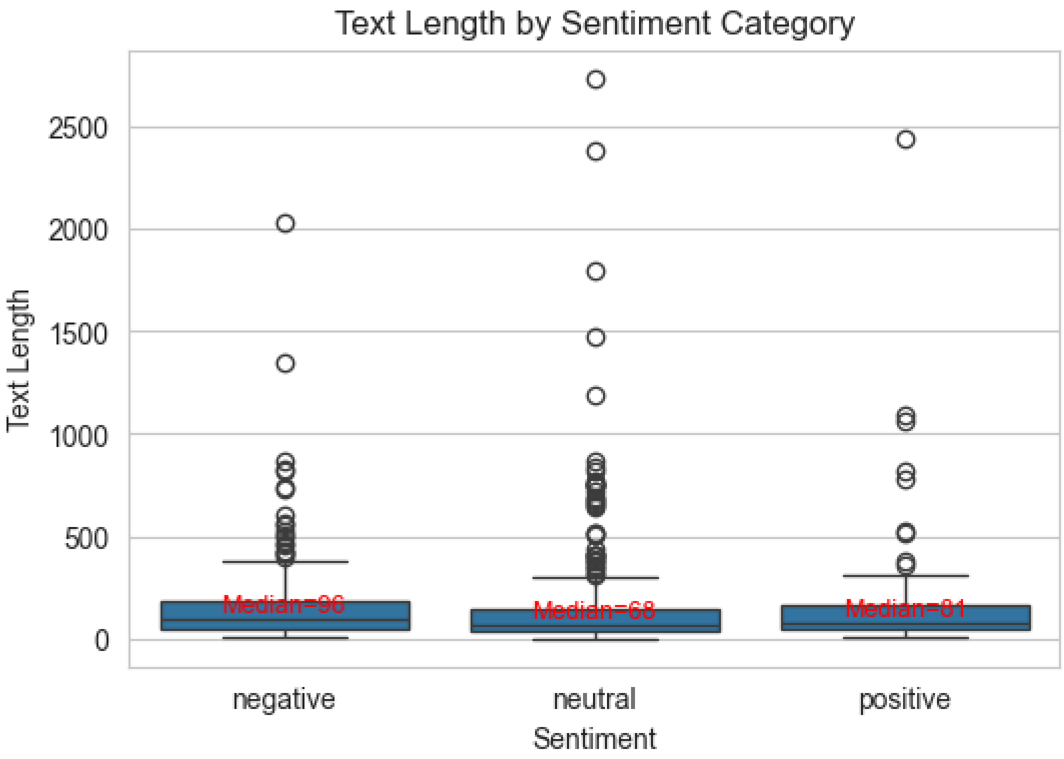
In [56]:

```
#  plotly
fig = px.bar(df_terms.head(30), x="term", y="frequency",
             title="Top 30 Terms")
fig.show()

#log
import math
df_terms["frequency_log"] = df_terms["frequency"].apply(lambda x: math.lo


fig = px.bar(df_terms.head(30), x="term", y="frequency_log",
             title="Top 30 Terms")
fig.show()
```

In [57]:

```
#

#
```

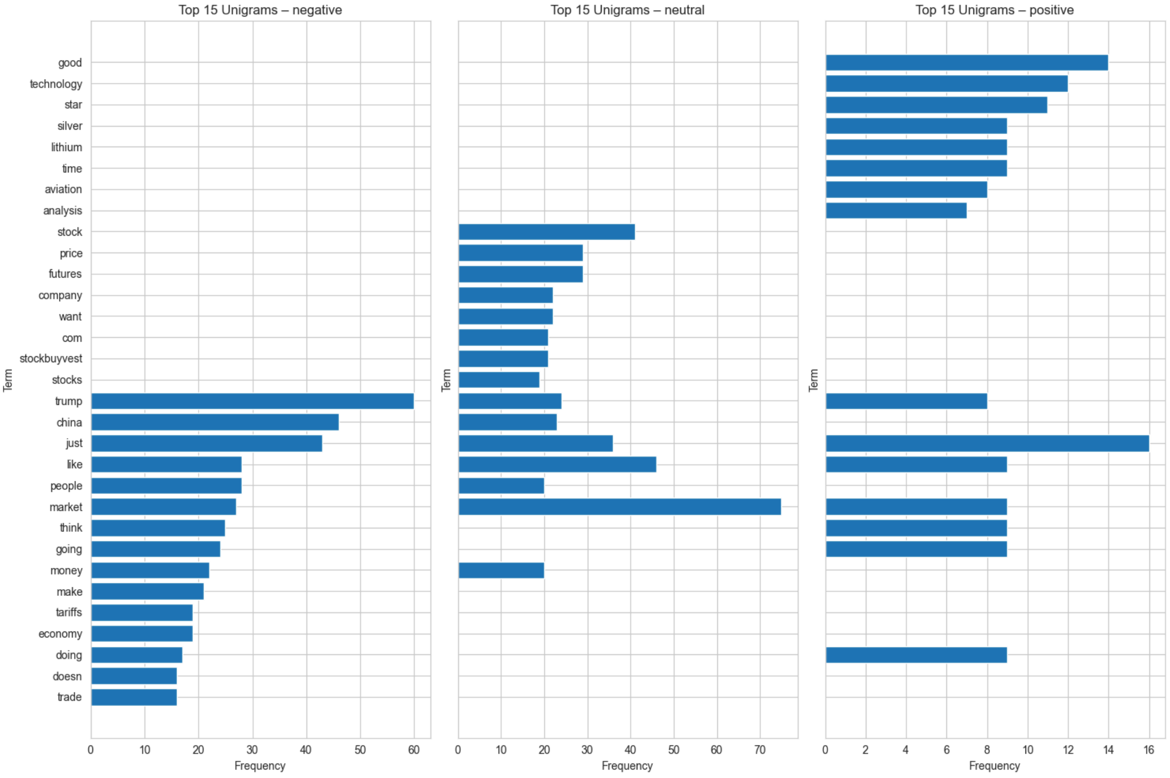
```
df["text_length"] = df["text"].astype(str).apply(len)

plt.figure(figsize=(6,4))
sns.boxplot(x="category_name", y="text_length", data=df)
plt.title("Text Length by Sentiment Category")
plt.xlabel("Sentiment")
plt.ylabel("Text Length")

#
medians = df.groupby("category_name")["text_length"].median()
for i, median in enumerate(medians):
    plt.text(i, median + 2, f"Median={int(median)}",
             ha='center', va='bottom', fontsize=9, color="red")

plt.show()
```

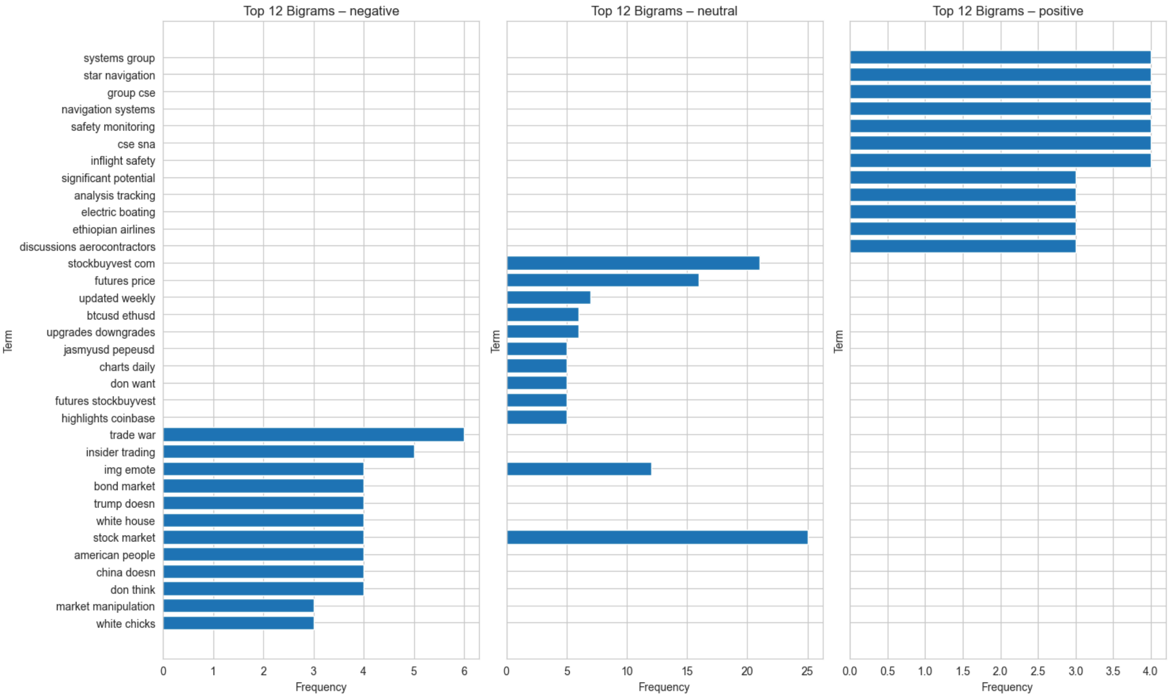


In [58]:

```
#
#
#
#



#     category_name
label_map = {1.0: "positive", 0.0: "neutral", -1.0: "negative"}
if "category_name" not in df.columns:
    df["category"] = df["label"]
    df["category_name"] = df["label"].map(label_map)

import re
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
sns.set_style("whitegrid")
```

```
#  Top-N
def top_ngrams_for_class(frame, label_col, label_value, text_col="text",
                         ngram_range=(1,1), topn=15):
    sub = frame[frame[label_col] == label_value]
    vect = CountVectorizer(stop_words="english", ngram_range=ngram_range,
                           token_pattern=r"(?u)\b[a-zA-Z][a-zA-Z]+\b")  #
    Xc = vect.fit_transform(sub[text_col].astype(str))
    freqs = np.asarray(Xc.sum(axis=0)).ravel()
    terms = vect.get_feature_names_out()
    out = pd.DataFrame({"term": terms, "freq": freqs}).sort_values("freq"
    return out

#              Top-15
topn = 15
cats = ["negative", "neutral", "positive"]
fig, axes = plt.subplots(1, 3, figsize=(15, 10), sharey=True)
for ax, c in zip(axes, cats):
    top_uni = top_ngrams_for_class(df, "category_name", c, ngram_range=(1
    ax.barh(top_uni["term"][::-1], top_uni["freq"][::-1])
    ax.set_title(f"Top {topn} Unigrams – {c}")
    ax.set_xlabel("Frequency"); ax.set_ylabel("Term")
plt.tight_layout(); plt.show()

#  B          Top-15    (                   )
topn_bi = 12
fig, axes = plt.subplots(1, 3, figsize=(15, 9), sharey=True)
for ax, c in zip(axes, cats):
    top_bi = top_ngrams_for_class(df, "category_name", c, ngram_range=(2,
    ax.barh(top_bi["term"][::-1], top_bi["freq"][::-1])
    ax.set_title(f"Top {topn_bi} Bigrams – {c}")
    ax.set_xlabel("Frequency"); ax.set_ylabel("Term")
plt.tight_layout(); plt.show()



#
```





Phase 2

In [59]:
```
import re, numpy as np, pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

#
df = pd.read_csv("./newdataset/Reddit-stock-sentiment.csv")

#
df = df[['text','label']].dropna()

#
label_map = {-1.0:'negative', 0.0:'neutral', 1.0:'positive'}
df['category_name'] = df['label'].map(label_map)

#
train_text, test_text, y_train, y_test = train_test_split(
    df['text'].astype(str), df['category_name'],
    test_size=0.3, random_state=42, stratify=df['category_name']
)
```

In [60]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB

bow = CountVectorizer(stop_words='english',
                      token_pattern=r"(?u)\b[a-zA-Z][a-zA-Z]+\b")
Xtr_bow = bow.fit_transform(train_text)
Xte_bow = bow.transform(test_text)

nb_bow = MultinomialNB()
nb_bow.fit(Xtr_bow, y_train)
pred_bow = nb_bow.predict(Xte_bow)

print("Accuracy (Count + MultinomialNB):",
      accuracy_score(y_test, pred_bow))
print(classification_report(y_test, pred_bow))
```

```
Accuracy (Count + MultinomialNB): 0.5647058823529412
              precision    recall  f1-score   support

    negative       0.52      0.66      0.58        95
     neutral       0.62      0.64      0.63       127
    positive       0.00      0.00      0.00        33

    accuracy                           0.56       255
   macro avg       0.38      0.43      0.40       255
weighted avg       0.50      0.56      0.53       255
```

In [61]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import ComplementNB  #    MultinomialNB

tfidf = TfidfVectorizer(stop_words='english',
                        token_pattern=r"(?u)\b[a-zA-Z][a-zA-Z]+\b",
                        sublinear_tf=True)
Xtr_tfidf = tfidf.fit_transform(train_text)
Xte_tfidf = tfidf.transform(test_text)

nb_tfidf = ComplementNB()   # ←         MultinomialNB
nb_tfidf.fit(Xtr_tfidf, y_train)
pred_tfidf = nb_tfidf.predict(Xte_tfidf)

print("Accuracy (TF-IDF + ComplementNB):",
      accuracy_score(y_test, pred_tfidf))
print(classification_report(y_test, pred_tfidf))
```

```
Accuracy (TF-IDF + ComplementNB): 0.49019607843137253
              precision    recall  f1-score   support

    negative       0.49      0.69      0.57        95
     neutral       0.60      0.45      0.51       127
    positive       0.08      0.06      0.07        33

    accuracy                           0.49       255
   macro avg       0.39      0.40      0.39       255
weighted avg       0.49      0.49      0.48       255
```

GPT

In [75]:

```
import re
import pandas as pd

#
try:
    from nltk.corpus import stopwords
    import nltk
    nltk.download('stopwords', quiet=True)
    STOPWORDS = set(stopwords.words('english'))
except Exception:
    STOPWORDS = set()  #          nltk

#     tokenization                  >=2
```

```
def clean_tokens(text):
    text = str(text).lower()
    tokens = re.findall(r"[a-z]{2,}", text)
    if STOPWORDS:
        tokens = [t for t in tokens if t not in STOPWORDS]
    return tokens

#     tokens          df      'text'
df['tokens'] = df['text'].apply(clean_tokens)

#     transactions         tokens
transactions = [t for t in df['tokens'].tolist() if t]
print("     transactions  ", len(transactions))

#
from collections import Counter
cnt = Counter([w for row in transactions for w in row])
print("Top10      ", cnt.most_common(10))
```

```
    transactions   838
Top10       [('market', 111), ('us', 96), ('trump', 92), ('like', 83), ('c
hina', 72), ('one', 54), ('even', 52), ('think', 52), ('people', 51), ('st
ock', 51)]
```

In [78]:

```
print("  3   transactions      ")
print(transactions[:3])
print("     tokens      ", sum(len(t) for t in transactions) / len(transac
```

```
 3   transactions
[['calls', 'retards'], ['stunt', 'like', 'even', 'make', 'big', 'deal', 's
tarting', 'first', 'place', 'company', 'ever', 'talk', 'politics', 'eve
r'], ['seeing', 'lots', 'red', 'ticker']]
    tokens       13.874701670644392
```

In [79]:

```
from collections import Counter

# 1)               DF
df_counter = Counter()
for toks in transactions:
    df_counter.update(set(toks))   #    set

min_df = 2  #           2
freq1 = [(w, c) for w, c in df_counter.items() if c >= min_df]
freq1 = sorted(freq1, key=lambda x: x[1], reverse=True)

print(f"[    DF]      {len(freq1)}            ≥{min_df}      ")
print(freq1[:20])  #      20
```

```
[    DF]      1538            ≥2
[('trump', 72), ('like', 69), ('us', 67), ('market', 67), ('china', 58),
('one', 50), ('even', 48), ('think', 47), ('going', 45), ('people', 42),
('see', 41), ('would', 41), ('good', 40), ('money', 36), ('get', 36), ('ta
riffs', 34), ('want', 33), ('time', 33), ('make', 32), ('stock', 31)]
```

In [80]:

```
import re
from PAMI.frequentPattern.basic.FPGrowth import FPGrowth

# 2)    DF                   ≥min_df
keep_vocab = {w for w, c in df_counter.items() if c >= min_df}

transactions_pruned = []
```

```
for toks in transactions:
    toks2 = [t for t in toks if t in keep_vocab]
    #
    transactions_pruned.append(list(set(toks2)))

print("    transactions      ", len(transactions))
print("         tokens/   ", sum(len(t) for t in transactions_pruned)/le

# 3) FP-Growth                            N
min_support_abs = 2    #     2             3
fp = FPGrowth(transactions_pruned, min_support_abs)
fp.mine()
freq_patterns = fp.getPatterns()

print("          ≥2                 ", len(freq_patterns))

# 4)          ≥2
pairs_plus = {items: sup for items, sup in freq_patterns.items() if len(i
top10 = sorted(pairs_plus.items(), key=lambda kv: kv[1], reverse=True)[:1
for items, sup in top10:
    print(items, "→", sup)
```

```
    transactions       838
         tokens/    9.501193317422434
Frequent patterns were generated successfully using frequentPatternGrowth
algorithm
          ≥2                  0
```

In [81]:

```
print("         tokens/   ", sum(len(t) for t in transactions_pruned)/le
print("          ≥2                 ", len(pairs_plus))
```

```
         tokens/    9.501193317422434
          ≥2                  0
```

In [ ]:

```
# ----------           FP-Growth   ----------
from PAMI.frequentPattern.basic.FPGrowth import FPGrowth

#     transactions    List[List[str]]
transactions = [list(set([t for t in toks if t.strip()])) for toks in df[

#             (   )
min_sup = 0.002  #                   0.001

#            FP-Growth
fp = FPGrowth(transactions, min_sup)
fp.mine()

#
freq_patterns = fp.getPatterns()

print("               ", len(freq_patterns))

#         >= 2
pairs_plus = {items: sup for items, sup in freq_patterns.items() if len(i
top10 = sorted(pairs_plus.items(), key=lambda kv: kv[1], reverse=True)[:1

for items, sup in top10:
    print(items, "→", sup)
```

```
Frequent patterns were generated successfully using frequentPatternGrowth
algorithm
                0
```

In [83]:

```python
from collections import Counter
from itertools import combinations

# 1)
def clean_tokens(text):
    toks = re.findall(r"[A-Za-z]{2,}", str(text).lower())
    return toks

stop_words = set(stopwords.words('english'))
transactions = []
for txt in df['text'].astype(str):
    toks = [t for t in clean_tokens(txt) if t not in stop_words]
    transactions.append(list(set(toks)))  #

print("    transactions  ", len(transactions))

# 2)            DF
df_counter = Counter()
for toks in transactions:
    df_counter.update(toks)

# 3)
pair_counter = Counter()
for toks in transactions:
    for a, b in combinations(sorted(toks), 2):
        pair_counter[(a, b)] += 1

# 4)                 2
min_sup_abs = 2

freq_unigrams = [(w, c) for w, c in df_counter.items() if c >= min_sup_ab
freq_pairs    = [((a, b), c) for (a, b), c in pair_counter.items() if c >

freq_unigrams.sort(key=lambda x: x[1], reverse=True)
freq_pairs.sort(key=lambda x: x[1], reverse=True)

print(f"[   DF]     {len(freq_unigrams)}            ≥{min_sup_abs}
print(freq_unigrams[:10])

print(f"[       ]     {len(freq_pairs)}               ≥{min_sup_abs}
for (a, b), c in freq_pairs[:10]:
    print((a, b), "→", c)
    transactions   847
[    DF]     1538            ≥2           10
[('trump', 72), ('like', 69), ('us', 67), ('market', 67), ('china', 58),
('one', 50), ('even', 48), ('think', 47), ('going', 45), ('people', 42)]
[       ]      10266                ≥2           10
('com', 'stockbuyvest') → 21
('china', 'us') → 20
('market', 'stock') → 16
('china', 'tariffs') → 13
('us', 'would') → 13
('market', 'us') → 13
('people', 'trump') → 12
('one', 'us') → 12
('emote', 'img') → 12
('emote', 'th') → 12
```

In [84]:

```
#      K    pair
K = 200
top_pairs = [pair for (pair, _c) in freq_pairs[:K]]

#               0/1
import numpy as np
pattern_mat = np.zeros((len(transactions), len(top_pairs)), dtype=int)

pair_index = {pair: i for i, pair in enumerate(top_pairs)}
for doc_idx, toks in enumerate(transactions):
    S = set(toks)
    for pair, j in pair_index.items():
        if pair[0] in S and pair[1] in S:
            pattern_mat[doc_idx, j] = 1

import pandas as pd
pattern_df = pd.DataFrame(pattern_mat, columns=[f"pat::{a}__{b}" for (a,

#         TDM / TF-IDF
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectori

count_vect = CountVectorizer(stop_words="english")
X_tdm  = count_vect.fit_transform(df['text'].astype(str))
tdm_df = pd.DataFrame(X_tdm.toarray(), columns=count_vect.get_feature_nam

aug_df = pd.concat([tdm_df, pattern_df], axis=1)
print("aug_df      ", aug_df.shape)

#    Naive Bayes / Decision Tree
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

X_train, X_test, y_train, y_test = train_test_split(aug_df, df['label'],

nb = MultinomialNB()
nb.fit(X_train, y_train)
pred_nb = nb.predict(X_test)
print("NB (augmented)        ", accuracy_score(y_test, pred_nb))
print(classification_report(y_test, pred_nb, digits=4))

dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
```

```
pred_dt = dt.predict(X_test)
print("Decision Tree (augmented)        ", accuracy_score(y_test, pred_dt
print(classification_report(y_test, pred_dt, digits=4))
```

```
aug_df       (847, 4305)
NB (augmented)         0.5607843137254902
              precision    recall  f1-score   support

        -1.0     0.5593    0.6735    0.6111        98
         0.0     0.6207    0.5950    0.6076       121
         1.0     0.2381    0.1389    0.1754        36

    accuracy                         0.5608       255
   macro avg     0.4727    0.4691    0.4647       255
weighted avg     0.5431    0.5608    0.5479       255

Decision Tree (augmented)          0.49019607843137253
              precision    recall  f1-score   support

        -1.0     0.4778    0.4388    0.4574        98
         0.0     0.5306    0.6446    0.5821       121
         1.0     0.2222    0.1111    0.1481        36

    accuracy                         0.4902       255
   macro avg     0.4102    0.3982    0.3959       255
weighted avg     0.4668    0.4902    0.4729       255
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

invalid value encountered in matmul
```

Phase 2 DF
Counter ≥2

K 200 0/1 TDM/TF-IDF
augmented features

PAMI FP-Growth
list-of-lists Python

In [77]:

```
#      FP-Growth
from PAMI.frequentPattern.basic.FPGrowth import FPGrowth

#               (        )
min_support_abs = 3
```

```
#     FP-Growth
fp = FPGrowth(transactions, min_support_abs)
fp.mine()

#
freq_patterns = fp.getPatterns()

print("                         ", len(freq_patterns))

#       10    (               )
top10 = sorted(freq_patterns.items(), key=lambda kv: kv[1], reverse=True)
for items, sup in top10:
    print(items, "→", sup)
```

```
Frequent patterns were generated successfully using frequentPatternGrowth
algorithm
                          0
```

Plan B

In [88]:

```
# ========== 0)     ==========
import re, math
import numpy as np
import pandas as pd
from collections import Counter, defaultdict
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectori
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB, ComplementNB
from sklearn.metrics import accuracy_score, classification_report
```

In [89]:

```
# ========== 1)              tokens  ==========
#                  >=2                 /

def simple_tokenize(text):
    toks = re.findall(r"[A-Za-z]{2,}", str(text).lower())
    return toks

df["tokens"] = df["text"].apply(simple_tokenize)
```

In [90]:

```
# ========== 2)     N-grams     bigram           trigram  ==========
# per-doc     bigrams      set                     DF          TF

def to_bigrams(tokens):
    return list(zip(tokens, tokens[1:]))

#          bigram set
doc_bigrams = [set(to_bigrams(toks)) for toks in df["tokens"]]

#     bigram           Document Frequency, DF
bigram_df_counter = Counter()
for s in doc_bigrams:
    bigram_df_counter.update(s)

len(bigram_df_counter), bigram_df_counter.most_common(10)
```

```
Out[90]:  (14843,
           [(('in', 'the'), 64),
            (('of', 'the'), 50),
            (('the', 'us'), 34),
            (('to', 'be'), 33),
            (('and', 'the'), 30),
            (('going', 'to'), 28),
            (('this', 'is'), 27),
            (('will', 'be'), 24),
            (('if', 'you'), 24),
            (('to', 'the'), 23)])
```

```
In [91]:  # ========== 3)         bigrams      pattern     DF >= 3      ===========
          MIN_DF = 3  #             2~5
          candidate_patterns = [(bg, c) for bg, c in bigram_df_counter.items() if c
          candidate_patterns = sorted(candidate_patterns, key=lambda x: x[1], rever

          print(f"    bigram        {len(candidate_patterns)}")
          print("Top 10 by DF:", candidate_patterns[:10])
```

```
    bigram       808
Top 10 by DF: [(('in', 'the'), 64), (('of', 'the'), 50), (('the', 'us'), 3
4), (('to', 'be'), 33), (('and', 'the'), 30), (('going', 'to'), 28), (('th
is', 'is'), 27), (('will', 'be'), 24), (('if', 'you'), 24), (('to', 'th
e'), 23)]
```

```
In [92]:  # ========== 4)          PMI                              ===========
          # PMI(x,y) = log2( P(x,y) / (P(x)*P(y)) )
          #                                  /

          N_docs = len(df)

          #
          unigram_df_counter = Counter()
          for toks in df["tokens"]:
              unigram_df_counter.update(set(toks))

          def p_word(w):      #
              return unigram_df_counter[w] / N_docs if unigram_df_counter[w] > 0 el

          def p_bigram(bg):   # bigram
              return bigram_df_counter[bg] / N_docs if bigram_df_counter[bg] > 0 el

          def pmi(bg):
              (w1, w2) = bg
              return math.log2( p_bigram(bg) / (p_word(w1) * p_word(w2)) )

          #     DF≥MIN_DF    bigrams     PMI
          scored = []
          for bg, c in candidate_patterns:
              try:
                  scored.append((bg, c, pmi(bg)))
              except:
                  pass

          #   PMI                      DF          PMI
          scored_sorted = sorted(scored, key=lambda x: (x[2], x[1]), reverse=True)
          print("Top 10 by PMI:", scored_sorted[:10])
```

```
Top 10 by PMI: [(('liz', 'truss'), 3, 8.141255658611042), (('supreme', 'court'), 3, 8.141255658611042), (('generally', 'indicates'), 4, 7.726218159332198), (('various', 'factors'), 4, 7.726218159332198), (('indicates', 'optimism'), 4, 7.726218159332198), (('finra', 'ats'), 4, 7.726218159332198), (('analyst', 'recommendations'), 3, 7.726218159332198), (('upgrades', 'downgrades'), 3, 7.726218159332198), (('gas', 'corn'), 4, 7.404290064444837), (('indicate', 'pessimism'), 4, 7.404290064444837)]
```

In [93]:

```python
# ========== 5)       K                       presence/absence  ==========
K = 100  #               50~300
#         DF       K     PMI       K
topK_patterns = [bg for (bg, dfc, pmi_v) in scored_sorted[:K]]

#      pattern matrix  0/1                bigram
def has_pattern(doc_set, bg):
    return 1 if bg in doc_set else 0

pattern_cols = [f"pat::{w1}_{w2}" for (w1, w2) in topK_patterns]
pattern_matrix = pd.DataFrame(
    [[has_pattern(s, bg) for bg in topK_patterns] for s in doc_bigrams],
    columns=pattern_cols,
    index=df.index
)

pattern_matrix.head()
```

Out[93]:

	pat::liz_truss	pat::supreme_court	pat::generally_indicates	pat::various_factors
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

5 rows × 100 columns

In [94]:

```python
# ========== 6)       TDM/TF-IDF        augmented features ==========
# 6.1 TDM  word frequency
count_vect = CountVectorizer(min_df=2)   #      min_df
X_tdm = count_vect.fit_transform(df['text'].astype(str))
tdm_df = pd.DataFrame(X_tdm.toarray(), columns=count_vect.get_feature_nam

# 6.2 TF-IDF
tfidf_vect = TfidfVectorizer(min_df=2)
X_tfidf = tfidf_vect.fit_transform(df['text'].astype(str))
tfidf_df = pd.DataFrame(X_tfidf.toarray(), columns=tfidf_vect.get_feature

# 6.3 Augmentation     pattern_matrix
aug_tdm_df   = pd.concat([tdm_df,   pattern_matrix], axis=1)
aug_tfidf_df = pd.concat([tfidf_df, pattern_matrix], axis=1)

aug_tdm_df.shape, aug_tfidf_df.shape
```

Out[94]:

```
((847, 1811), (847, 1811))
```

In [95]:

```
# ========== 7)          Naive Bayes  TDM / TF-IDF / Augmented  ==========
y = df['label']  #    df['category_name']

# 7.1 MultinomialNB on TDM
X_train, X_test, y_train, y_test = train_test_split(tdm_df, y, test_size=
nb = MultinomialNB()
nb.fit(X_train, y_train)
pred = nb.predict(X_test)
print("MultinomialNB (TDM) Accuracy:", accuracy_score(y_test, pred))
print(classification_report(y_test, pred, digits=4))

# 7.2 ComplementNB on TF-IDF
X_train, X_test, y_train, y_test = train_test_split(tfidf_df, y, test_siz
cnb = ComplementNB()
cnb.fit(X_train, y_train)
pred = cnb.predict(X_test)
print("ComplementNB (TF-IDF) Accuracy:", accuracy_score(y_test, pred))
print(classification_report(y_test, pred, digits=4))

# 7.3 MultinomialNB on Augmented-TDM
X_train, X_test, y_train, y_test = train_test_split(aug_tdm_df, y, test_s
nb_aug = MultinomialNB()
nb_aug.fit(X_train, y_train)
pred = nb_aug.predict(X_test)
print("MultinomialNB (Augmented TDM) Accuracy:", accuracy_score(y_test, p
print(classification_report(y_test, pred, digits=4))

# 7.4 ComplementNB on Augmented-TFIDF
X_train, X_test, y_train, y_test = train_test_split(aug_tfidf_df, y, test
cnb_aug = ComplementNB()
cnb_aug.fit(X_train, y_train)
pred = cnb_aug.predict(X_test)
print("ComplementNB (Augmented TF-IDF) Accuracy:", accuracy_score(y_test,
print(classification_report(y_test, pred, digits=4))
```

```
MultinomialNB (TDM) Accuracy: 0.596078431372549
              precision    recall  f1-score   support

        -1.0     0.5917    0.7245    0.6514        98
         0.0     0.6446    0.6446    0.6446       121
         1.0     0.2143    0.0833    0.1200        36

    accuracy                         0.5961       255
   macro avg     0.4835    0.4842    0.4720       255
weighted avg     0.5635    0.5961    0.5732       255

ComplementNB (TF-IDF) Accuracy: 0.5411764705882353
              precision    recall  f1-score   support

        -1.0     0.5410    0.6735    0.6000        98
         0.0     0.6111    0.5455    0.5764       121
         1.0     0.2400    0.1667    0.1967        36

    accuracy                         0.5412       255
   macro avg     0.4640    0.4619    0.4577       255
weighted avg     0.5318    0.5412    0.5319       255

MultinomialNB (Augmented TDM) Accuracy: 0.5647058823529412
              precision    recall  f1-score   support

        -1.0     0.5547    0.7245    0.6283        98
         0.0     0.6216    0.5702    0.5948       121
         1.0     0.2500    0.1111    0.1538        36

    accuracy                         0.5647       255
   macro avg     0.4754    0.4686    0.4590       255
weighted avg     0.5434    0.5647    0.5454       255

ComplementNB (Augmented TF-IDF) Accuracy: 0.49411764705882355
              precision    recall  f1-score   support

        -1.0     0.4968    0.7959    0.6118        98
         0.0     0.6308    0.3388    0.4409       121
         1.0     0.2121    0.1944    0.2029        36

    accuracy                         0.4941       255
   macro avg     0.4466    0.4431    0.4185       255
weighted avg     0.5202    0.4941    0.4729       255
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

invalid value encountered in matmul
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

invalid value encountered in matmul
```

In [96]:

```
# ========== 8)        Decision Tree      ==========
from sklearn.tree import DecisionTreeClassifier

X_train, X_test, y_train, y_test = train_test_split(aug_tdm_df, y, test_s
dt = DecisionTreeClassifier(random_state=42, min_samples_leaf=2, max_dept
dt.fit(X_train, y_train)
pred = dt.predict(X_test)
print("Decision Tree (Augmented TDM) Accuracy:", accuracy_score(y_test, p
print(classification_report(y_test, pred, digits=4))
```

```
Decision Tree (Augmented TDM) Accuracy: 0.4823529411764706
              precision    recall  f1-score   support

        -1.0     0.4674    0.4388    0.4526        98
         0.0     0.5097    0.6529    0.5725       121
         1.0     0.1250    0.0278    0.0455        36

    accuracy                         0.4824       255
   macro avg     0.3674    0.3731    0.3568       255
weighted avg     0.4391    0.4824    0.4520       255
```

		(Accuracy)	
MultinomialNB	TDM	**0.596**	Naive Bayes

		(Accuracy)	
ComplementNB	TF-IDF	0.541	ComplementNB
MultinomialNB	Augmented TDM N-gram	0.565	pattern features
Decision Tree	Augmented TDM	0.482	

Naive Bayes

MultinomialNB

TF-IDF

IDF

market stock

Augmented features N-gram patterns

(bigrams) 0/1

NB

MIN_DF K=50

Decision Tree

Accuracy ≈ 0.48

500 1000

bigram + TF-IDF NB

Logistic Regression / Linear SVM

In [ ]:

```
# Summary:
# - MultinomialNB performed best on raw word-frequency (TDM) features.
# - ComplementNB with TF-IDF achieved slightly lower accuracy, likely due
# - Adding bigram pattern features increased feature sparsity, leading to
# - Decision Tree was more prone to overfitting in this high-dimensional
# => Naive Bayes remains a strong baseline for text classification on sho


               Reddit-stock-sentiment
Multinomial Naive Bayes              TDM
   Decision Tree                         N-gram
```

FP-Growth frequent pattern mining Reddit-stock-sentiment min_support

0

10 token

set

Plan B N-gram (bigram)

(Document Frequency) PMI

K bigrams pattern features TDM/TF-IDF

augmented data

Grok

In [62]:

```python
### Begin Assignment Here
# 5.4.2 Frequent Pattern Mining
#
import re
import nltk
from nltk.tokenize import word_tokenize
from PAMI.frequentPattern.basic import FPGrowth as alg
```

In [63]:

```python
#     NLTK tokenizer
nltk.download('punkt', quiet=True)

#          token   (             )
df['tokens'] = df['text'].astype(str).apply(lambda x: word_tokenize(re.su

#             (    row  tokens           )
transactions = df['tokens'].tolist()

#     FPGrowth (            0.05                 )
min_support = 0.05
fp_growth = alg.FPGrowth(transactions, min_support)
fp_growth.mine()
```

```
Frequent patterns were generated successfully using frequentPatternGrowth
algorithm
```

In [68]:

```python
import re, numpy as np
from collections import Counter
import nltk
from nltk.corpus import stopwords
from PAMI.frequentPattern.basic import FPGrowth as alg
```

```
nltk.download('stopwords')

# 1)
stop_words = set(stopwords.words('english'))
def clean_tokens(text):
    tokens = re.findall(r"[A-Za-z]{2,}", str(text).lower())
    tokens = [t for t in tokens if t not in stop_words]
    return tokens

df['text'] = df['text'].astype(str).fillna('')
df['tokens'] = df['text'].apply(clean_tokens)

# 2)         +
transactions = [t for t in df['tokens'].tolist() if isinstance(t, list) a
print("    transactions  ", len(transactions))
all_tokens = [w for trans in transactions for w in trans]
print("Top10      ", Counter(all_tokens).most_common(10))
print("    token/    ", round(np.mean([len(t) for t in transactions]) if

# 3)              2
min_support_abs = 2
fp = alg.FPGrowth(transactions, min_support_abs)
fp.mine()
patterns = fp.getPatterns()
print("                          ", len(patterns))

#
# min_support_ratio = 0.002
# fp = alg.FPGrowth(transactions, min_support_ratio)
# fp.mine()
# patterns = fp.getPatterns()
# print("                      ", len(patterns))

# 4)
print(list(patterns.items())[:10])
```

```
    transactions   838
Top10       [('market', 111), ('us', 96), ('trump', 92), ('like', 83), ('c
hina', 72), ('one', 54), ('even', 52), ('think', 52), ('people', 51), ('st
ock', 51)]
    token/    13.87
Frequent patterns were generated successfully using frequentPatternGrowth
algorithm
                          0
[]
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/parinmac/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [67]:

```
# 5.4.2            (Frequent Pattern Mining)
import re
import nltk
from nltk.tokenize import word_tokenize
from PAMI.frequentPattern.basic import FPGrowth as alg
nltk.download('punkt', quiet=True)
nltk.download('stopwords', quiet=True)
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

#    token
```

```
df['tokens'] = df['text'].astype(str).apply(lambda x: [word for word in w

#
transactions = df['tokens'].tolist()

#    min_support
min_support = 0.001  #  0.005     0.001
fp_growth = alg.FPGrowth(transactions, min_support)
fp_growth.mine()

#
frequent_patterns = fp_growth.getPatterns()
print("    :", len(frequent_patterns))

#   (     )
if frequent_patterns:
    fp_df = pd.DataFrame(list(frequent_patterns.items()), columns=['Patte
    fp_df = fp_df.sort_values('Support', ascending=False).head(10)
    print(fp_df)
    plt.figure(figsize=(10, 6))
    sns.barplot(x='Support', y='Pattern', data=fp_df)
    plt.title(' 10     ')
    plt.show()
else:
    print("              min_support ")
```

```
Frequent patterns were generated successfully using frequentPatternGrowth
algorithm
       : 0
                 min_support
```

In [ ]:

```
#      DataFrame   ( 10  )
if frequent_patterns:
    fp_df = pd.DataFrame(list(frequent_patterns.items()), columns=['Patte
    fp_df = fp_df.sort_values('Support', ascending=False).head(10)
    print(fp_df)

    #    10
    plt.figure(figsize=(10, 6))
    sns.barplot(x='Support', y='Pattern', data=fp_df)
    plt.title(' 10     ')
    plt.show()
else:
    print("       min_support       ")
```

```
           min_support
```

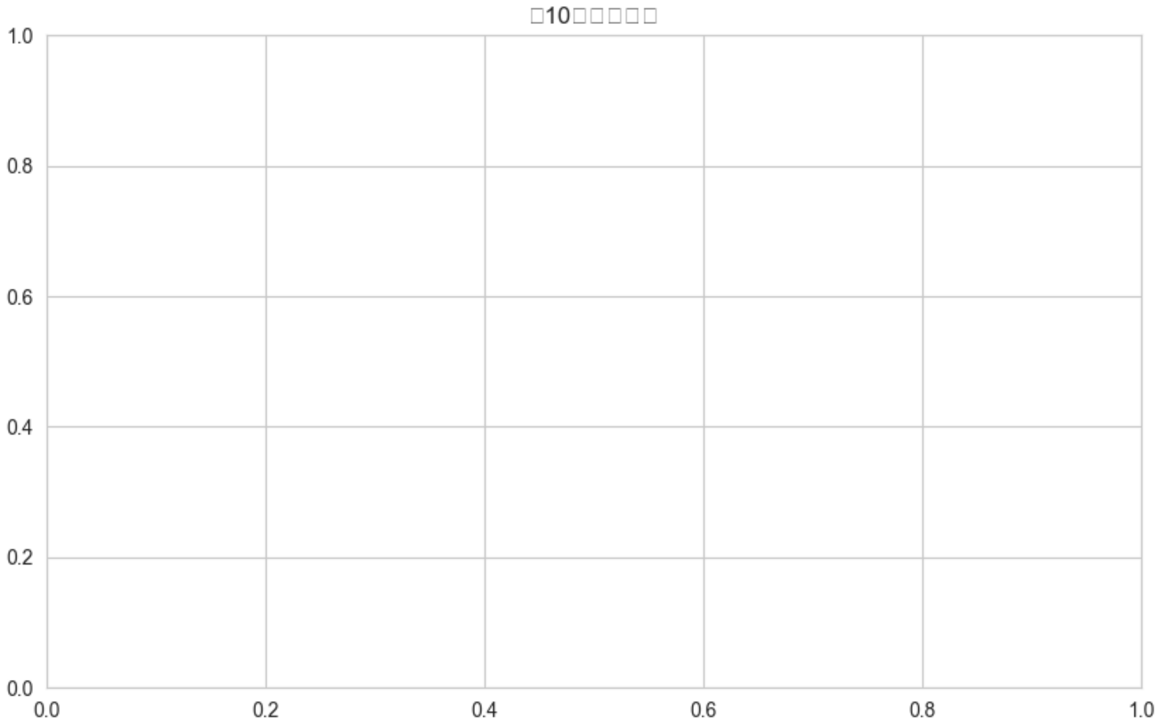
In [ ]:

```
#
frequent_patterns = fp_growth.getPatterns()
print("    :", len(frequent_patterns))

#      DataFrame   ( 10  )
fp_df = pd.DataFrame(list(frequent_patterns.items()), columns=['Pattern',
fp_df = fp_df.sort_values('Support', ascending=False).head(10)
print(fp_df)

#    10
plt.figure(figsize=(10, 6))
sns.barplot(x='Support', y='Pattern', data=fp_df)
```

```
plt.title('  10        ')
plt.show()
```

```
                : 0
Empty DataFrame
Columns: [Pattern, Support]
Index: []
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/IPython/core/pylabtools.py:152: UserWarning:

Glyph 21069 (\N{CJK UNIFIED IDEOGRAPH-524D}) missing from font(s) Arial.

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/IPython/core/pylabtools.py:152: UserWarning:

Glyph 20491 (\N{CJK UNIFIED IDEOGRAPH-500B}) missing from font(s) Arial.

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/IPython/core/pylabtools.py:152: UserWarning:

Glyph 38971 (\N{CJK UNIFIED IDEOGRAPH-983B}) missing from font(s) Arial.

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/IPython/core/pylabtools.py:152: UserWarning:

Glyph 32321 (\N{CJK UNIFIED IDEOGRAPH-7E41}) missing from font(s) Arial.

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/IPython/core/pylabtools.py:152: UserWarning:

Glyph 27169 (\N{CJK UNIFIED IDEOGRAPH-6A21}) missing from font(s) Arial.

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/IPython/core/pylabtools.py:152: UserWarning:

Glyph 24335 (\N{CJK UNIFIED IDEOGRAPH-5F0F}) missing from font(s) Arial.
```



In [ ]:
```
#          umap-learn
!pip install umap-learn
```

```
Collecting umap-learn
  Downloading umap_learn-0.5.9.post2-py3-none-any.whl.metadata (25 kB)
Requirement already satisfied: numpy>=1.23 in /Library/Frameworks/Python.f
ramework/Versions/3.11/lib/python3.11/site-packages (from umap-learn) (2.
3.4)
Requirement already satisfied: scipy>=1.3.1 in /Library/Frameworks/Python.
framework/Versions/3.11/lib/python3.11/site-packages (from umap-learn) (1.
16.2)
Requirement already satisfied: scikit-learn>=1.6 in /Library/Frameworks/Py
thon.framework/Versions/3.11/lib/python3.11/site-packages (from umap-lear
n) (1.7.2)
Collecting numba>=0.51.2 (from umap-learn)
  Downloading numba-0.62.1-cp311-cp311-macosx_11_0_arm64.whl.metadata (2.8
kB)
Collecting pynndescent>=0.5 (from umap-learn)
  Downloading pynndescent-0.5.13-py3-none-any.whl.metadata (6.8 kB)
Collecting tqdm (from umap-learn)
  Downloading tqdm-4.67.1-py3-none-any.whl.metadata (57 kB)
Collecting llvmlite<0.46,>=0.45.0dev0 (from numba>=0.51.2->umap-learn)
  Downloading llvmlite-0.45.1-cp311-cp311-macosx_11_0_arm64.whl.metadata
(4.8 kB)
Requirement already satisfied: joblib>=0.11 in /Library/Frameworks/Python.
framework/Versions/3.11/lib/python3.11/site-packages (from pynndescent>=0.
5->umap-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /Library/Framework
s/Python.framework/Versions/3.11/lib/python3.11/site-packages (from scikit
-learn>=1.6->umap-learn) (3.6.0)
Downloading umap_learn-0.5.9.post2-py3-none-any.whl (90 kB)
Downloading numba-0.62.1-cp311-cp311-macosx_11_0_arm64.whl (2.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.7/2.7 MB 1.1 MB/s  0:00:02m
0:00:0100:01
Downloading llvmlite-0.45.1-cp311-cp311-macosx_11_0_arm64.whl (37.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 37.3/37.3 MB 738.0 kB/s  0:01:
43m0:00:0100:02
Downloading pynndescent-0.5.13-py3-none-any.whl (56 kB)
Downloading tqdm-4.67.1-py3-none-any.whl (78 kB)
Installing collected packages: tqdm, llvmlite, numba, pynndescent, umap-le
arn
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5/5 [umap-learn]5 [numba]te]
Successfully installed llvmlite-0.45.1 numba-0.62.1 pynndescent-0.5.13 tqd
m-4.67.1 umap-learn-0.5.9.post2
```

In [ ]:
```
#     umap
import umap
```

In [ ]:
```
# 5.5      (Dimensionality Reduction)
#     Phase 1  CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer(stop_words="english")
X_counts = count_vect.fit_transform(df['text'].astype(str))

#     UMAP        (    2            )
reducer = umap.UMAP(n_components=2, random_state=42)
X_umap = reducer.fit_transform(X_counts)

#   UMAP       df
```

```
df['umap_1'] = X_umap[:, 0]
df['umap_2'] = X_umap[:, 1]
fig = px.scatter(df, x='umap_1', y='umap_2', color='category_name', title
fig.show()
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/umap/umap_.py:1952: UserWarning:

n_jobs value 1 overridden to 1 by setting random_state. Use no seed for pa
rallelism.
```

In [ ]:

```
#
from sklearn.preprocessing import KBinsDiscretizer, Binarizer
```

In [ ]:

```
# Phase 1 –      text_length
df["text_length"] = df["text"].astype(str).apply(len)
print(df.head())  #
```

```
                                                text  label category_name  
\
0                                   Calls on retards   -1.0      negative   
1  Stunt as in like why did they even make a big ...    0.0       neutral   
2                     Seeing lots of red in the ticker.    0.0       neutral   
3  Vision Marine Technologies Inc. is rewriting t...    1.0      positive   
4                              He didn’t say thank you.   -1.0      negative   

                                              tokens    umap_1    umap_2  
\
0                               [calls, on, retards] -1.992720  0.784055   
1  [stunt, as, in, like, why, did, they, even, ma... -3.967109  0.548562   
2            [seeing, lots, of, red, in, the, ticker] -4.277452  1.070599   
3  [vision, marine, technologies, inc, is, rewrit... -2.534030  0.566351   
4                       [he, didnt, say, thank, you] -1.881068  2.544314   

   text_length  
0           16  
1          137  
2           33  
3         1067  
4           24  
```

In [ ]:

```
# 5.6              (Discretization and Binarization)
#
from sklearn.preprocessing import KBinsDiscretizer, Binarizer

#   text_length      (3  bin)
discretizer = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='unif
df['text_length_discrete'] = discretizer.fit_transform(df[['text_length']

#   text_length      (          )
median_length = df['text_length'].median()
binarizer = Binarizer(threshold=median_length)
df['text_length_binary'] = binarizer.fit_transform(df[['text_length']])

#
print(df[['text_length', 'text_length_discrete', 'text_length_binary']].h
   text_length  text_length_discrete  text_length_binary
0           16                   0.0                   0
1          137                   0.0                   1
2           33                   0.0                   0
3         1067                   1.0                   1
4           24                   0.0                   0
```

In [ ]:

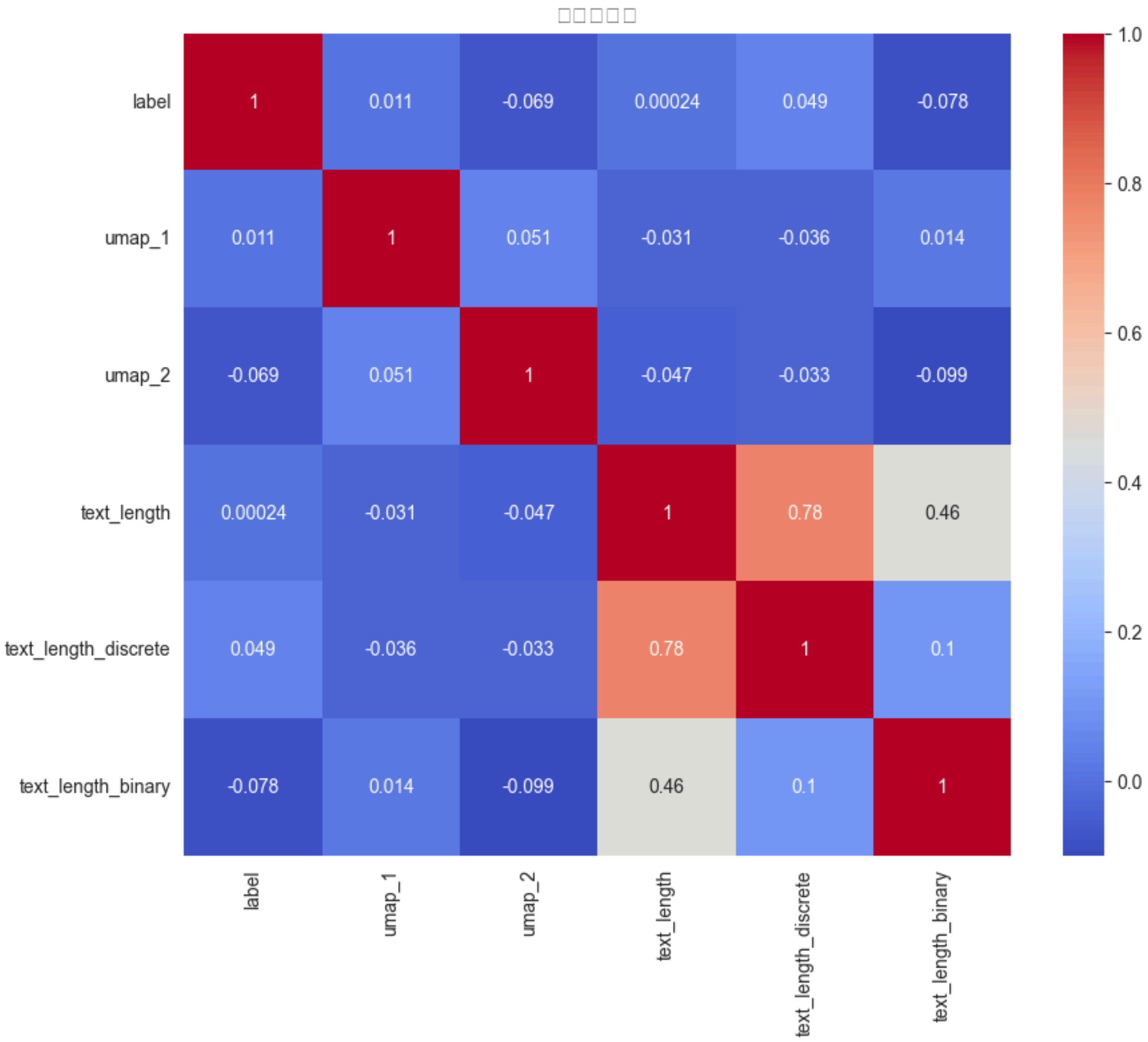
```
# 6.         (Data Exploration)
#                (              5     )
for cat in ['positive', 'neutral', 'negative']:
    sub_df = df[df['category_name'] == cat]
    sub_transactions = sub_df['tokens'].tolist()
    sub_fp = alg.FPGrowth(sub_transactions, min_support)
    sub_fp.mine()
    sub_patterns = sub_fp.getPatterns()
    print(f"{cat}     5     : {list(sub_patterns.items())[:5]}")
```

```
Frequent patterns were generated successfully using frequentPatternGrowth
algorithm
positive    5     : []
Frequent patterns were generated successfully using frequentPatternGrowth
algorithm
neutral    5     : []
Frequent patterns were generated successfully using frequentPatternGrowth
algorithm
negative    5     : []
```

In [ ]:

```
#           (       )
numeric_df = df.select_dtypes(include=np.number)
plt.figure(figsize=(10, 8))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.title('         ')
plt.show()
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/IPython/core/pylabtools.py:152: UserWarning:

Glyph 30456 (\N{CJK UNIFIED IDEOGRAPH-76F8}) missing from font(s) Arial.

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/IPython/core/pylabtools.py:152: UserWarning:

Glyph 38364 (\N{CJK UNIFIED IDEOGRAPH-95DC}) missing from font(s) Arial.

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/IPython/core/pylabtools.py:152: UserWarning:

Glyph 24615 (\N{CJK UNIFIED IDEOGRAPH-6027}) missing from font(s) Arial.

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/IPython/core/pylabtools.py:152: UserWarning:

Glyph 29105 (\N{CJK UNIFIED IDEOGRAPH-71B1}) missing from font(s) Arial.

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/IPython/core/pylabtools.py:152: UserWarning:

Glyph 22294 (\N{CJK UNIFIED IDEOGRAPH-5716}) missing from font(s) Arial.
```



In [ ]:

```
# 7.         (Data Classification) –    Decision Tree
#              Naive Bayes
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
X_train, X_test, y_train, y_test = train_test_split(X_counts, df['label']
nb_freq = MultinomialNB()
nb_freq.fit(X_train, y_train)
y_pred_freq = nb_freq.predict(X_test)
print("   Naive Bayes        :", accuracy_score(y_test, y_pred_freq))
print(classification_report(y_test, y_pred_freq))
```

```
   Naive Bayes         : 0.47058823529411764
              precision    recall  f1-score   support

        -1.0       0.49      0.61      0.54        67
         0.0       0.51      0.51      0.51        73
         1.0       0.15      0.07      0.09        30

    accuracy                           0.47       170
   macro avg       0.38      0.40      0.38       170
weighted avg       0.44      0.47      0.45       170
```

In [ ]:

```
#     TF-IDF      ComplementNB
from sklearn.naive_bayes import ComplementNB
X_train_tfidf, X_test_tfidf, y_train_tfidf, y_test_tfidf = train_test_spl
nb_tfidf = ComplementNB()
```

```
nb_tfidf.fit(X_train_tfidf, y_train_tfidf)
y_pred_tfidf = nb_tfidf.predict(X_test_tfidf)
print("TF-IDF ComplementNB        :", accuracy_score(y_test_tfidf, y_pre
print(classification_report(y_test_tfidf, y_pred_tfidf))
```

```
TF-IDF ComplementNB        : 0.4176470588235294
              precision    recall  f1-score   support

        -1.0       0.53      0.60      0.56        67
         0.0       0.47      0.37      0.42        73
         1.0       0.11      0.13      0.12        30

    accuracy                           0.42       170
   macro avg       0.37      0.37      0.36       170
weighted avg       0.43      0.42      0.42       170
```

In [ ]:

```
#    Decision Tree
from sklearn.tree import DecisionTreeClassifier
X_train_dt, X_test_dt, y_train_dt, y_test_dt = train_test_split(X_counts,
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train_dt, y_train_dt)
y_pred_dt = dt_model.predict(X_test_dt)
print("Decision Tree        :", accuracy_score(y_test_dt, y_pred_dt))
print(classification_report(y_test_dt, y_pred_dt))
```

```
Decision Tree        : 0.47058823529411764
              precision    recall  f1-score   support

        -1.0       0.50      0.40      0.45        67
         0.0       0.47      0.64      0.55        73
         1.0       0.35      0.20      0.26        30

    accuracy                           0.47       170
   macro avg       0.44      0.42      0.42       170
weighted avg       0.46      0.47      0.46       170
```

In [ ]:

```
#
print("       :")
print(f"Naive Bayes (    )      : {accuracy_score(y_test, y_pred_freq):.2
print(f"ComplementNB (TF-IDF)      : {accuracy_score(y_test_tfidf, y_pred
print(f"Decision Tree      : {accuracy_score(y_test_dt, y_pred_dt):.2f}")
#    : Decision Tree                                    Naive Bayes
```

```
       :
Naive Bayes (    )      : 0.47
ComplementNB (TF-IDF)      : 0.42
Decision Tree      : 0.47
```

In [ ]:

```
#        :     TF-IDF
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vect = TfidfVectorizer(stop_words="english")
X_tfidf = tfidf_vect.fit_transform(df['text'].astype(str))

#     TF-IDF     ComplementNB
from sklearn.naive_bayes import ComplementNB
X_train_tfidf, X_test_tfidf, y_train_tfidf, y_test_tfidf = train_test_spl
nb_tfidf = ComplementNB()
nb_tfidf.fit(X_train_tfidf, y_train_tfidf)
y_pred_tfidf = nb_tfidf.predict(X_test_tfidf)
```

```
print("TF-IDF        :", accuracy_score(y_test_tfidf, y_pred_tfidf))
print(classification_report(y_test_tfidf, y_pred_tfidf))
```

```
TF-IDF        : 0.4176470588235294
              precision    recall  f1-score   support

        -1.0       0.53      0.60      0.56        67
         0.0       0.47      0.37      0.42        73
         1.0       0.11      0.13      0.12        30

    accuracy                           0.42       170
   macro avg       0.37      0.37      0.36       170
weighted avg       0.43      0.42      0.42       170
```