IST 332: natural Language Processing

Final Project: Analyzing AI-Related Job Anxiety and Future Optimism on Reddit

Github Link to access Reports and Code Notebook: https://github.com/dolmarawat/NLP-332-Final-Project

Task 1: Create Corpus

```
Start coding or generate with AI.
```

Import NLTK libraries

```
!pip install nltk spacy emoji
!python -m spacy download en_core_web_sm
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.12/dis
Requirement already satisfied: spacy in /usr/local/lib/python3.12/di
Collecting emoji
  Downloading emoji-2.15.0-py3-none-any.whl.metadata (5.7 kB)
Requirement already satisfied: click in /usr/local/lib/python3.12/di
Requirement already satisfied: joblib in /usr/local/lib/python3.12/d
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/pyt
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dis
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/l
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/l
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/loc
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/l
Requirement already satisfied: thinc<8.4.0,>=8.3.4 in /usr/local/lib
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/li
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local
```

```
Requirement already satisfied: weasel<0.5.0,>=0.4.2 in /usr/local/li
Requirement already satisfied: typer-slim<1.0.0,>=0.3.0 in /usr/loca
Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/pythc
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/d
Requirement already satisfied: setuptools in /usr/local/lib/python3.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/pyt
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/
Requirement already satisfied: pydantic-core==2.41.4 in /usr/local/l
Requirement already satisfied: typing-extensions>=4.14.1 in /usr/loc
Requirement already satisfied: typing-inspection>=0.4.2 in /usr/loca
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/loca
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/pythor
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/
Requirement already satisfied: blis<1.4.0,>=1.3.0 in /usr/local/lib/
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/loca
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in /usr/lc
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in /usr/loca
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/pyt
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/di
Downloading emoji-2.15.0-py3-none-any.whl (608 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 608.4/608.4 kB 19.3 MB/s
Installing collected packages: emoji
Successfully installed emoji-2.15.0
Collecting en-core-web-sm==3.8.0
  Downloading https://github.com/explosion/spacy-models/releases/dow
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 63.9 MB/s
✔ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart P
order to load all the package's dependencies. You can do this by sel
'Restart kernel' or 'Restart runtime' option.
```

Importing word lemmatizer for creating POS tag types

```
import nltk
from string import punctuation
nltk.download('stopwords')
nltk.download('punkt')

from nltk.corpus import stopwords
import re
def preprocessing(tokens):
  tokens = [token.lower() for token in tokens] # lowercasing

  tokens = [token for token in tokens if not token.isdigit()] # rem

  tokens = [token for token in tokens if token not in punctuation]

  # Remove tokens that are entirely punctuation (single or multiple
  tokens = [token for token in tokens if not re.fullmatch(r"[^\w]+"

  mystopwords = set(stopwords.words("english")) # use english stopw
  tokens = [token for token in tokens if token not in mystopwords]

  tokens = [token for token in tokens if len(token)>=3] # remove to

  return tokens
```

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

```
from nltk.stem import SnowballStemmer

snow_stemmer = SnowballStemmer(language='english')  # set the langu
#stem each word in the token list of each review
def snowball_Stemmer(token_col):
  """ stem each word in the specfical col """
  snowball_words = []
  for w in token_col:
    x = snow_stemmer.stem(w)
    snowball_words.append(x)
  return snowball_words
```

Mounting drive and dataset

```
from google.colab import drive
drive.mount('/content/Drive',  force_remount=True)
```

```
Mounted at /content/Drive
```

```
import pandas as pd
df = pd.read_csv('/content/Drive/MyDrive/IST332AIReddit/reddit_revi
df.head()
```

	text	label	type
0	I GOT THE JOB!! F*** MY OLD MANAGER!!! I've ha...	CAREER_ANXIETY	post
1	congrats! just a word of caution not to tell t...	CAREER_ANXIETY	comment
2	Congrats on the pay raise! Did you study anyth...	CAREER_ANXIETY	comment
3	update to us how you break the news to your ma...	CAREER_ANXIETY	comment
4	Congrats. You can do the bare minimum now and ...	CAREER_ANXIETY	comment

Inspecting data

```
# Inspect
print(df.shape)
print(df.columns.tolist())
df.head(3)
```

```
(10000, 3)
['text', 'label', 'type']
```

	text	label	type
0	I GOT THE JOB!! F*** MY OLD MANAGER!!! I've ha...	CAREER_ANXIETY	post
1	congrats! just a word of caution not to tell t...	CAREER_ANXIETY	comment
2	Congrats on the pay raise! Did you study anyth...	CAREER_ANXIETY	comment

Task 2: Text Preprocessing

Expanded contractions “don’t” → “do not” This helps models understand negation clearly. Tokenize Split into individual words. Normalize lowercase remove digits remove punctuation remove stopwords keep words with ≥3 characters Applied Stemming A rough vocabulary reduction. Applied Lemmatization A precise vocabulary reduction. Create a fully cleaned text column For downstream modeling.

```
!pip install contractions
import contractions
```

```
Collecting contractions
  Downloading contractions-0.1.73-py2.py3-none-any.whl.metadata (1.2
Collecting textsearch>=0.0.21 (from contractions)
  Downloading textsearch-0.0.24-py2.py3-none-any.whl.metadata (1.2 k
Collecting anyascii (from textsearch>=0.0.21->contractions)
  Downloading anyascii-0.3.3-py3-none-any.whl.metadata (1.6 kB)
Collecting pyahocorasick (from textsearch>=0.0.21->contractions)
  Downloading pyahocorasick-2.2.0-cp312-cp312-manylinux_2_17_x86_64.
Downloading contractions-0.1.73-py2.py3-none-any.whl (8.7 kB)
Downloading textsearch-0.0.24-py2.py3-none-any.whl (7.6 kB)
Downloading anyascii-0.3.3-py3-none-any.whl (345 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 345.1/345.1 kB 21.1 MB/s
Downloading pyahocorasick-2.2.0-cp312-cp312-manylinux_2_17_x86_64.ma
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 114.9/114.9 kB 11.7 MB/s
Installing collected packages: pyahocorasick, anyascii, textsearch,
Successfully installed anyascii-0.3.3 contractions-0.1.73 pyahocoras
```

```
import nltk
from string import punctuation
from nltk.corpus import stopwords
import re

def preprocessing(tokens):
  tokens = [token.lower() for token in tokens] # lowercasing
  tokens = [token for token in tokens if not token.isdigit()] # rem
  tokens = [token for token in tokens if token not in punctuation]
  # Remove tokens that are entirely punctuation (single or multiple
  tokens = [token for token in tokens if not re.fullmatch(r"[^\w]+"
  mystopwords = set(stopwords.words("english")) # use english stopw
  tokens = [token for token in tokens if token not in mystopwords]
  tokens = [token for token in tokens if len(token)>=3] # remove to
  return tokens
```

```
df['Review_Expanded'] = df['text']
```

```
from nltk.tokenize import wordpunct_tokenize

# Get a sample expanded text from the DataFrame for demonstration
expanded_text_sample = df['Review_Expanded'].iloc[0]

# tokenization
tokens = wordpunct_tokenize(expanded_text_sample)

# run basic preprocessing
pre_proc_text = preprocessing(tokens)

print("Original expanded text sample:", expanded_text_sample)
print("Preprocessed text:", pre_proc_text)
print("Length of preprocessed text:", len(pre_proc_text))
```

```
Original expanded text sample: I GOT THE JOB!! F*** MY OLD MANAGER!!
Preprocessed text: ['got', 'job', 'old', 'manager', 'deal', 'extreme
Length of preprocessed text: 46
```

```
df.columns
```

```
Index(['text', 'label', 'type', 'Review_Expanded'], dtype='object')
```

```
print(df.columns.tolist())
```

```
['text', 'label', 'type', 'Review_Expanded']
```

```
# Preprocess the first 10 reviews
preprocessed_reviews = []
for i in range(0, 10):  # Iterate through the first 10 rows (index
    # Get the already expanded text from the DataFrame
    expanded_text = df['Review_Expanded'].iloc[i]
    # Tokenize and preprocess the expanded text
    tokens = wordpunct_tokenize(expanded_text)
    preprocessed_tokens = preprocessing(tokens)
    preprocessed_reviews.append(preprocessed_tokens)

# Display the preprocessed reviews
for i, review_tokens in enumerate(preprocessed_reviews):
    print(f"Review {i+1}: {review_tokens}\n")
```

```
Review 1: ['got', 'job', 'old', 'manager', 'deal', 'extremely', 'tox

Review 2: ['congrats', 'word', 'caution', 'tell', 'manager', 'anyone

Review 3: ['congrats', 'pay', 'raise', 'study', 'anything', 'system'

Review 4: ['update', 'break', 'news', 'manager']

Review 5: ['congrats', 'bare', 'minimum', 'fully', 'cut', 'start', '

Review 6: ['hoping', 'thrive', 'one', 'positive', 'people', 'around'

Review 7: ['congratulations', 'left', 'incredibly', 'toxic', 'manage

Review 8: ['need', 'learn', 'deal', 'toxic', 'managers', 'need', 'wo

Review 9: ['leaving', 'never', 'seeing', 'toxic', 'manager', 'one',

Review 10: ['congrats', 'man', 'awesome', 'hear', 'sounds', 'like',
```

```
import nltk, contractions
from nltk import word_tokenize
nltk.download('punkt', quiet=True)
nltk.download('punkt_tab', quiet=True) # Added to download the miss

df["raw_tokens"] = df["text"].fillna("").apply(
    lambda t: word_tokenize(contractions.fix(t))
)
```

```
import re, string
from nltk.corpus import stopwords
nltk.download('stopwords', quiet=True)

stop = set(stopwords.words("english"))
extra_stop = {"sep"}  # month slipped in from dates; add more if yo
stop |= extra_stop

def normalize(tokens):
    out = []
    for w in tokens:
        w = (w or "").lower()
        w = (w.replace("’","'").replace("“",'"').replace("”",'"').r
        w = w.strip(string.punctuation)       # remove leading/trai
        if not w or len(w) < 2:    # drop empties and 1-char tokens
            continue
        if w in stop:              # drop stopwords (and 'sep')
            continue
        if re.search(r"\d", w):    # drop tokens containing digits
            continue
        # drop tokens that are still all punctuation-like
        if all(ch in string.punctuation for ch in w):
            continue
        out.append(w)
    return out

df["clean_tokens"] = df["raw_tokens"].apply(normalize)
```

```
from nltk import FreqDist
corpus_clean = [w for review in df["clean_tokens"] for w in review]
fdist = FreqDist(corpus_clean)

print("Top 10 tokens (normalized):")
for tok, cnt in fdist.most_common(25):
    print(tok, cnt)
```

```
Top 10 tokens (normalized):
like 2032
ai 2028
people 1797
get 1647
job 1592
would 1586
work 1376
time 1204
one 1125
years 1117
even 1012
going 989
company 925
know 906
good 894
make 874
think 855
need 780
much 776
got 768
still 763
companies 761
jobs 747
also 703
new 689
```

```
df['Review_Expanded'] = df['text'].apply(lambda x:contractions.fix(
df[["text", "Review_Expanded"]].head(10)
```

	text	Review_Expanded
0	I GOT THE JOB!! F*** MY OLD MANAGER!!! I've ha...	i got the job!! f*** my old manager!!! i have ...
1	congrats! just a word of caution not to tell t...	congrats! just a word of caution not to tell t...
2	Congrats on the pay raise! Did you study anyth...	congrats on the pay raise! did you study anyth...
3	update to us how you break the news to your ma...	update to us how you break the news to your ma...
4	Congrats. You can do the bare minimum now and ...	congrats. you can do the bare minimum now and ...
5	Hoping you thrive at this one and have positiv...	hoping you thrive at this one and have positiv...
6	Congratulations! I just left an incredibly tox...	congratulations! i just left an incredibly tox...
7	You need to learn to deal with toxic managers....	you need to learn to deal with toxic managers....
8	Leaving and never seeing a toxic manager again...	leaving and never seeing a toxic manager again...

```
df.describe()
```

	text	label	type	Review
count	10000	10000	10000	
unique	9987	2	2	
top	![gif](giphy|XvjC06Gh9lhfZNBNIM)	CAREER_ANXIETY	comment	(giphy|xvjc06gl
freq	3	5500	9817	

```
import spacy
nlp = spacy.load("en_core_web_sm", disable=["parser", "ner"])
```

```
%%time
df['Review_Tokens'] = df['Review_Expanded'].apply(lambda review:[tok
```

```
CPU times: user 58.3 s, sys: 264 ms, total: 58.6 s
Wall time: 1min 12s
```

```
texts = df["Review_Expanded"].astype(str).tolist()
```

```
df[['text', 'Review_Expanded', "Review_Tokens"]].head(10)
```

	text	Review_Expanded	Review_Tokens
0	I GOT THE JOB!! F*** MY OLD MANAGER!!! I've ha...	i got the job!! f*** my old manager!!! i have ...	[i, got, the, job, !, !, f, *, *, *, my, old, ...
1	congrats! just a word of caution not to tell t...	congrats! just a word of caution not to tell t...	[congrats, !, just, a, word, of, caution, not,...
2	Congrats on the pay raise! Did you study anyth...	congrats on the pay raise! did you study anyth...	[congrats, on, the, pay, raise, !, did, you, s...
3	update to us how you break the news to your ma...	update to us how you break the news to your ma...	[update, to, us, how, you, break, the, news, t...
4	Congrats. You can do the bare minimum now and ...	congrats. you can do the bare minimum now and ...	[congrats, ., you, can, do, the, bare, minimum...
5	Hoping you thrive at this one and have positiv...	hoping you thrive at this one and have positiv...	[hoping, you, thrive, at, this, one, and, have...
6	Congratulations! I just left an incredibly tox...	congratulations! i just left an incredibly tox...	[congratulations, !, i, just, left, an, incred...
7	You need to learn to deal with toxic managers....	you need to learn to deal with toxic managers....	[you, need, to, learn, to, deal, with, toxic, ...

Normalisation with Stemmer

```
from nltk.stem import SnowballStemmer

snow_stemmer = SnowballStemmer(language='english')  # set the langu
#stem each word in the token list of each review
def snowball_Stemmer(token_col):
  """ stem each word in the specfical col """
  snowball_words = []
  for w in token_col:
    x = snow_stemmer.stem(w)
    snowball_words.append(x)
  return snowball_words
```

```
%%time
df['Review_Stemmed'] = df['Review_Tokens'].apply(snowball_Stemmer)
```

```
CPU times: user 5.02 s, sys: 13.8 ms, total: 5.04 s
Wall time: 5.12 s
```

```
df['Review_Stemmed'].head(10)
```

	Review_Stemmed
0	[i, got, the, job, !, !, f, *, *, *, my, old, ...
1	[congrat, !, just, a, word, of, caution, not, ...
2	[congrat, on, the, pay, rais, !, did, you, stu...
3	[updat, to, us, how, you, break, the, news, to...
4	[congrat, ., you, can, do, the, bare, minimum,...
5	[hope, you, thrive, at, this, one, and, have, ...
6	[congratul, !, i, just, left, an, incred, toxi...
7	[you, need, to, learn, to, deal, with, toxic, ...
8	[leav, and, never, see, a, toxic, manag, again...
9	[congrat, man, !, that, is, awesom, to, hear, ...

dtype: object

Normalisation through Lemmas

```
# Function to return just lemmas from text
# spacy carry the string than tokens
def spacy_lemmatizer(text):
  doc = nlp(text)
  return [token.lemma_ for token in doc if not token.is_space]
```

```
 #Apply spacy lemmtization
%%time
df['Review_Lemma'] = df['Review_Expanded'].apply(spacy_lemmatizer)
df['Review_Lemma'].head(3)
```

```
CPU times: user 46.7 s, sys: 303 ms, total: 47 s
Wall time: 47.3 s
```

	Review_Lemma
0	[I, get, the, job, !, !, f, *, *, *, my, old, ...
1	[congrat, !, just, a, word, of, caution, not, ...
2	[congrat, on, the, pay, raise, !, do, you, stu...

dtype: object

```
df['Review_Lemma_Final'] = df['Review_Lemma'].apply(preprocessing)
df['Review_Cleaned'] = df['Review_Lemma_Final'].apply(lambda row: "
df['Review_Cleaned'].head()
```

	Review_Cleaned
0	get job old manager deal extremely toxic manag...
1	congrat word caution tell manager anyone team ...
2	congrat pay raise study anything system design...
3	update break news manager
4	congrat bare minimum fully cut start new gig

dtype: object

```
#data cleaning for unnecessary jargon on reddit comments

import re

def clean_reddit(text):
    text = re.sub(r'http\S+', ' ', text)                # remove U
    text = re.sub(r'www\.\S+', ' ', text)               # remove U
    text = re.sub(r'amp', ' ', text)                    # remove "
    text = re.sub(r'reddit\S*', ' ', text)              # remove r
    text = re.sub(r'webp\S*', ' ', text)                # remove i
    text = re.sub(r'png|jpg|jpeg|gif', ' ', text)       # remove i
    text = re.sub(r'\s+', ' ', text)                    # normaliz
    return text.strip()


df["Review_Cleaned"] = df["Review_Cleaned"].apply(clean_reddit)
```

```
print("Shape of the DataFrame before removing rows with empty lemma

# Remove rows where 'Review_Lemma_Final' is an empty list
df_cleaned = df[df['Review_Lemma_Final'].apply(lambda x: len(x) > 0

print("Shape of the DataFrame after removing rows with empty lemma
```

```
Shape of the DataFrame before removing rows with empty lemma lists:
Shape of the DataFrame after removing rows with empty lemma lists: (
```

```
# Display the first few rows of the cleaned reviews
print("First 5 cleaned reviews:")
display(df_cleaned['Review_Cleaned'].head())
```

```
First 5 cleaned reviews:
```

	Review_Cleaned
0	get job old manager deal extremely toxic manag...
1	congrat word caution tell manager anyone team ...
2	congrat pay raise study anything system design...
3	update break news manager
4	congrat bare minimum fully cut start new gig

dtype: object

```
df_cleaned['Review_Length'] = df_cleaned['Review_Cleaned'].apply(la

print("Descriptive statistics for token lengths:")
print(df_cleaned['Review_Length'].describe())

print("\nFirst 5 cleaned reviews with their lengths:")
display(df_cleaned[['Review_Cleaned', 'Review_Length']].head(20))
```

```
Descriptive statistics for token lengths:
count    10000.00000
mean        22.15770
std         30.61643
min          0.00000
25%          7.00000
50%         12.00000
75%         25.00000
```

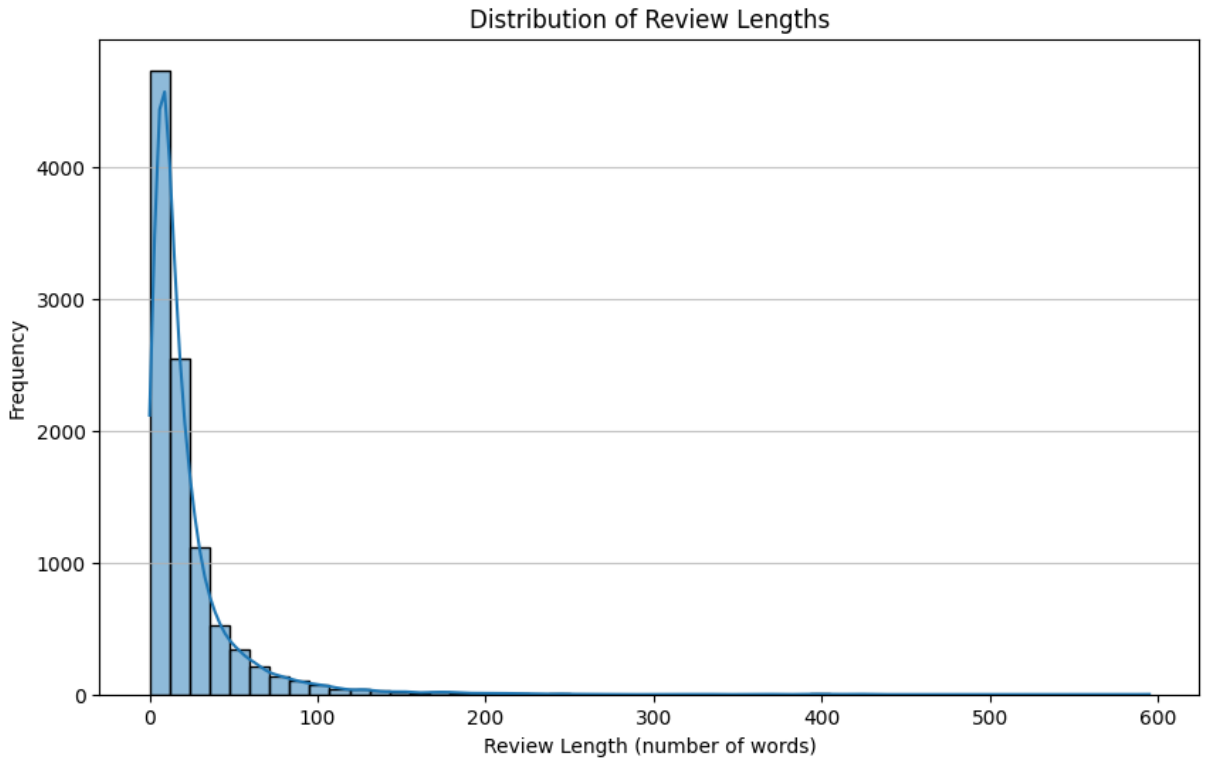
```
max        595.00000
Name: Review_Length, dtype: float64

First 5 cleaned reviews with their lengths:
```

	Review_Cleaned	Review_Length
0	get job old manager deal extremely toxic manag...	46
1	congrat word caution tell manager anyone team ...	16
2	congrat pay raise study anything system design...	8
3	update break news manager	4
4	congrat bare minimum fully cut start new gig	8
5	hop thrive one positive people around	6
6	congratulation leave incredibly toxic manager ...	13
7	need learn deal toxic manager need work weeken...	9
8	leave never see toxic manager one liberating f...	8
9	congrat man awesome hear sound like deserve wo...	66
10	feel publicly humiliate staff exclude staff me...	57
11	remember job actually start burn bridge yet	7
12	work everyone know apply	4
13	new job fuck old manager	5
14	live goddamn dream congratulation	4
15	really want able say soon congrat man	7
16	dude huge congrat seriously walk place drain l...	22
17	congrat take time celebrate recalibrate let bu...	10
18	congratulation get land something well huge wi...	70
19	lot venting congrat offer	4

```
import nltk
from nltk.probability import FreqDist

# Ensure 'all_words' is available (if not, it would need to be re-r
# For safety, re-create all_words from df_cleaned['Review_Cleaned']
all_words = ' '.join(df_cleaned['Review_Cleaned']).split()

fdist = FreqDist(all_words)
fdist.most_common(25)
```

```
[('get', 3106),
 ('job', 2359),
 ('work', 2107),
 ('like', 2086),
 ('people', 1812),
 ('year', 1702),
 ('company', 1695),
 ('make', 1597),
 ('would', 1588),
 ('time', 1431),
 ('one', 1246),
 ('good', 1240),
 ('know', 1164),
 ('think', 1147),
 ('use', 1122),
 ('well', 1119),
 ('thing', 1101),
 ('need', 1064),
 ('say', 1025),
 ('even', 1012),
 ('want', 923),
 ('take', 915),
 ('see', 913),
 ('look', 864),
 ('pay', 791)]
```

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.histplot(df_cleaned['Review_Length'], bins=50, kde=True)
plt.title('Distribution of Review Lengths')
plt.xlabel('Review Length (number of words)')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.show()
```



Task 3: Data Understanding

Finding and exploring context of words

```
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
True
```

```
from nltk.text import Text

# 'corpus_clean' was already created in a previous step (cell 08X3X
# It contains tokens processed with a less restrictive length filte
reviews_text_nltk = Text(corpus_clean)

# What does AI often occur with?
reviews_text_nltk.concordance('ai')
```

```
Displaying 25 of 2028 matches:
ing indeed grown tiresome given state ai bubble say sorry guys usa e
g developers hiring discount thinking ai insanity would stopped mark
i insanity would stopped market crash ai race china continue oracle
hinese us government full bailout big ai players fear software engir
ory everyone picked wrong major india ai need organized labor moveme
drop bucket people still hiring crazy ai related stuff happened bigg
teresting see far companies push idea ai efficiency gains enable low
 quick note anyone new blame anything ai offshoring people think abs
ads vibe code way college avoid using ai assistance much possible ha
e salary including devs also increase ai software jobs becoming scar
even networking taking big hit wonder ai impacted graduating numbers
ople find cut people able game system ai also many probably find hum
domain question job existence ongoing ai advancement millions freshe
ion never really learned code without ai know troubleshoot evaluate
omy flourishing start ups powered bit ai think ai going canvas peopl
ishing start ups powered bit ai think ai going canvas people use pla
back year start cutting needed talent ai gratz dude saw posts like l
lenge even andrew ng saying overhyped ai causing harm cs degree stil
ght would useful field worried impact ai careers especially new grad
google brian coursera harm overhyping ai tl dr frontier model progre
 email titled dilemma late contribute ai author gave permission shar
paring college worried time graduates ai good meaningful work left c
ence encouraged work hard learn build ai conversation struck example
versation struck example harmful hype ai gt gt even though llms hand
general set tasks previous iterations ai technology compared humans
```

```
# What words appear with scared in similar context?
reviews_text_nltk.similar('scared')
```

```
people dream place would help point always likely figure clueless tr
curious force cringe trillionaire terrifying falsifiable
```

```
# common_contexts examines the contexts shared by two or more words
reviews_text_nltk.common_contexts(['ai', 'scared'])
```

```
No common contexts were found
```

```
# common_contexts examines the contexts shared by two or more words
reviews_text_nltk.common_contexts(['ai', 'job'])
```

```
ai_ai even_really use_search
```

```
from nltk.text import Text

# flatten raw tokens and lower-case
all_raw_lower = [w.lower() for review in df["raw_tokens"] for w in

text_obj = Text(all_raw_lower)
```

```
print("Words appearing in contexts similar to 'scared':")
text_obj.similar("scared", num=20)
```

```
Words appearing in contexts similar to 'scared':
hard good doing applying willing fun used going out how trying time
there able done kind lucky tired valuable expected
```

Exploratory data analysis

- Conduct exploratory data analysis to understand your text distribution, label

balance, and content quality.

Firstly, we'll be calculating Lexical Diversity

```
# Return total number of tokens in a text object
print(len(texts))
# Recall set items do not allow dupliates. Hence, below code return
print(len(set(texts)))
```

```
10000
9987
```

```
#Lexical diversity is the total number of tokens over unique tokens
def lexical_diversity(text):
    """
    A measure of the lexical richness of the text
    """
    return len(text)/len(set(text))
```

```
# percentage of a word in a text
def percentage(word_count, total):
    """
    Compute what percentage of the text is taken up by a specific w
    """
    return 100* word_count/total
```

```
# calculate lexical diversity of cleaned corpus
print("Lexical diversity of cleaned corpus:", lexical_diversity(cor
# calculate percentage of 'scared' in the cleaned corpus
# Ensure 'scared' is lowercased to match corpus_clean tokens
scared_count = corpus_clean.count('scared')
total_words = len(corpus_clean)
print("Percentage of 'scared' in cleaned corpus:", percentage(scare
```

```
Lexical diversity of cleaned corpus: 11.895519262981574
Percentage of 'scared' in cleaned corpus: 0.012761163818140214
```

```
import nltk, contractions, re, string
from nltk.corpus import stopwords
from nltk import word_tokenize

stop = set(stopwords.words("english"))

def normalize(tokens):
    cleaned = []
    for w in tokens:
        wl = w.lower().strip(string.punctuation)
        if not wl or len(wl) < 2:
            continue
        if wl in stop:
            continue
        if re.search(r"\d", wl):
            continue
        cleaned.append(wl)
    return cleaned

df["raw_tokens"] = df["text"].fillna("").apply(
    lambda t: word_tokenize(contractions.fix(t))
)
df["clean_tokens"] = df["raw_tokens"].apply(normalize)
```

```
def lexical_diversity(tokens):
    return (len(set(tokens)) / len(tokens)) if tokens else 0

rows = []
for i in range(min(10, len(df))):
    raw_len   = len(df.loc[i, "raw_tokens"])
    clean_len = len(df.loc[i, "clean_tokens"])
    lexdiv    = round(lexical_diversity(df.loc[i, "clean_tokens"]),
    rows.append([i+1, raw_len, clean_len, lexdiv])

import pandas as pd
table_df = pd.DataFrame(rows, columns=[
    "Review",
    "Review Length (# of tokens from raw text)",
    "Cleaned Review Length (# of tokens after normalization)",
    "Lexical Diversity"
])
table_df
```

	Review	Review Length (# of tokens from raw text)	Cleaned Review Length (# of tokens after normalization)	Lexical Diversity
0	1	112	46	0.8913
1	2	40	17	1.0000
2	3	17	8	1.0000
3	4	11	5	1.0000
4	5	20	8	1.0000
5	6	12	6	1.0000
6	7	32	14	1.0000
7	8	21	9	0.8889
8	9	16	8	1.0000
9	10	160	65	0.8769

```
def lexical_diversity(tokens):
    return (len(set(tokens)) / len(tokens)) if tokens else 0

rows = []
for i in range(min(10, len(df))):
    raw_len   = len(df.loc[i, "raw_tokens"])
    clean_len = len(df.loc[i, "clean_tokens"])
    lexdiv    = round(lexical_diversity(df.loc[i, "clean_tokens"]),
    rows.append([i+1, raw_len, clean_len, lexdiv])

import pandas as pd
table_df = pd.DataFrame(rows, columns=[
    "Review",
    "Review Length (# of tokens from raw text)",
    "Cleaned Review Length (# of tokens after normalization)",
    "Lexical Diversity"
])
table_df
```

	Review	Review Length (# of tokens from raw text)	Cleaned Review Length (# of tokens after normalization)	Lexical Diversity
0	1	112	46	0.8913
1	2	40	17	1.0000
2	3	17	8	1.0000
3	4	11	5	1.0000
4	5	20	8	1.0000
5	6	12	6	1.0000
6	7	32	14	1.0000
7	8	21	9	0.8889
8	9	16	8	1.0000
9	10	160	65	0.8769

Plot frequency distribuition

```
from nltk import FreqDist
```

```
def preprocessing(tokens):
    tokens = [token.lower() for token in tokens] # lowercasing
    tokens = [token for token in tokens if not token.isdigit()] # r
    tokens = [token for token in tokens if token not in punctuation
    mystopwords = set(stopwords.words("english")) # use english stc
    tokens = [token for token in tokens if token not in mystopwords
    tokens = [token for token in tokens if len(token)>=3] # remove
    return tokens
```

```
# Clean the raw text
all_reviews_text = ' '.join(df_cleaned['Review_Cleaned']) # Join th
tokens = wordpunct_tokenize(all_reviews_text)
# Run basic preprocessing
reviews_cleaned = preprocessing (tokens)
len(reviews_cleaned)
```

```
220401
```

```
# Use FreqDis() to find the frequency disctibution for each token
fdist_reviews= FreqDist(reviews_cleaned)
# Check fdist1, which is a dictionary data type that has key:value
fdist_reviews
```

```
FreqDist({'get': 3106, 'job': 2361, 'work': 2110, 'like': 2086,
'people': 1812, 'year': 1706, 'company': 1697, 'make': 1598,
'would': 1588, 'time': 1432, ...})
```

```
# count the total number of tokens in a corpus, which is the same a
fdist_reviews.N()
```

```
220401
```

```
#Count number of 'scared' appears
fdist_reviews['scared']
```

```
27
```

```
# determine relevant frequency of a token in a corpus
# this is very important because we also will remove low frequency
fdist_reviews.freq('scared')
```

```
0.00012250398137939484
```

```
#Create a list of vocabulary that includes all the keys or unique t
vocabulary_reviews = list(fdist_reviews.keys())
# Show the 20 most frequent words
vocabulary_reviews[:30]
```

```
['get',
 'job',
 'old',
 'manager',
 'deal',
 'extremely',
 'toxic',
 'month',
 'use',
 'personal',
 'insult',
 'make',
 'work',
 'weekend',
 'put',
 'zombie',
 'project',
 'study',
 'ass',
 'interview',
 'finally',
 'offer',
 'today',
 'role',
 'big',
 'tech',
 'way',
 'line',
 'actually',
 'want']
```

```
import re, string
from nltk.corpus import stopwords
nltk.download('stopwords', quiet=True)

stop = set(stopwords.words("english"))
extra_stop = {"sep"}  # month slipped in from dates; add more if yo
stop |= extra_stop

def normalize(tokens):
    out = []
    for w in tokens:
        w = (w or "").lower()
        w = (w.replace("’","'").replace("“",'"').replace("”",'"').r
        w = w.strip(string.punctuation)       # remove leading/trai
        if not w or len(w) < 2:    # drop empties and 1-char tokens
            continue
        if w in stop:              # drop stopwords (and 'sep')
            continue
        if re.search(r"\d", w):    # drop tokens containing digits
            continue
        # drop tokens that are still all punctuation-like
        if all(ch in string.punctuation for ch in w):
            continue
        out.append(w)
    return out
```

```
from nltk import FreqDist
corpus_clean = [w for review in df["clean_tokens"] for w in review]
fdist = FreqDist(corpus_clean)

print("Top 20 tokens (normalized):")
for tok, cnt in fdist.most_common(20):
    print(tok, cnt)
```

```
Top 20 tokens (normalized):
like 2031
ai 2027
people 1795
get 1647
job 1592
would 1586
work 1376
time 1202
one 1124
years 1114
even 1012
going 989
company 924
know 906
good 894
make 874
think 854
need 780
much 776
got 768
```

```
# sort by count (descending)
fdist_reviews_list = sorted(fdist_reviews.items(), key=lambda x: x[

# take top 20 (keep as a list to preserve order)
top20_tokens =fdist.most_common(20)

# print token + count
for token, count in top20_tokens:
    print(token, count)
```

```
like 2031
ai 2027
people 1795
get 1647
job 1592
would 1586
work 1376
time 1202
one 1124
years 1114
even 1012
going 989
company 924
know 906
good 894
make 874
think 854
need 780
much 776
got 768
```

```
for word, cnt in sorted(fdist_reviews.items(), key=lambda kv: kv[1]
    if cnt > 12:
        print(word, cnt)
```

```
get 3106
job 2361
work 2110
like 2086
people 1812
year 1706
company 1697
make 1598
would 1588
time 1432
one 1248
good 1242
know 1164
think 1147
use 1125
well 1119
```

```
thing 1101
need 1067
say 1025
even 1013
want 924
take 916
see 915
look 864
pay 792
much 777
still 763
feel 741
way 740
also 703
new 701
really 681
try 677
start 675
something 668
find 652
day 644
tech 637
month 632
lay 623
could 619
back 610
come 609
bad 589
lot 581
give 576
right 551
keep 542
hire 528
money 527
layoff 526
interview 521
never 520
happen 507
many 497
end 494
long 490
experience 482
ask 473
```

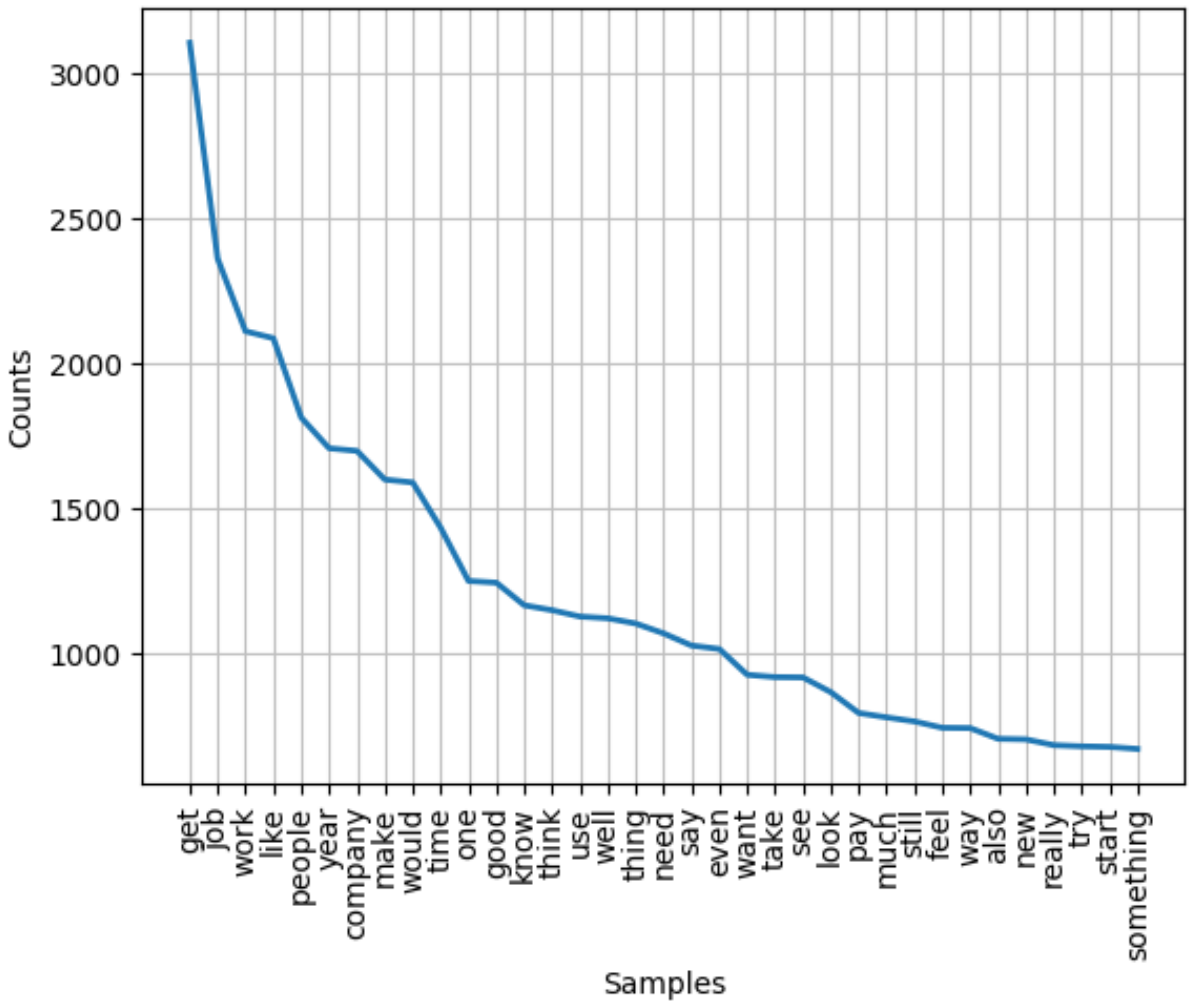
```
for word, cnt in fdist_reviews.most_common():
    if cnt <= 12:
        break
    print(word, cnt)
```

```
get 3106
job 2361
work 2110
like 2086
```

```
people 1812
year 1706
company 1697
make 1598
would 1588
time 1432
one 1248
good 1242
know 1164
think 1147
use 1125
well 1119
thing 1101
need 1067
say 1025
even 1013
want 924
take 916
see 915
look 864
pay 792
much 777
still 763
feel 741
way 740
also 703
new 701
really 681
try 677
start 675
something 668
find 652
day 644
tech 637
month 632
lay 623
could 619
back 610
come 609
bad 589
lot 581
give 576
right 551
keep 542
hire 528
money 527
layoff 526
interview 521
never 520
happen 507
many 497
end 494
long 490
```

```
experience 482
```

```
# plot the top 35 most frequent tokens
fdist_reviews.plot(35, cumulative=False)
plt.show()
```



Further Data Understanding

```
# Length stats by label
length_stats = df_cleaned.groupby("label")["Review_Length"].describ
display(length_stats)
```

label	count	mean	std	min	25%	50%	75%	max
CAREER_ANXIETY	5500.0	26.50	33.45	0.0	8.0	16.0	32.0	595.0
FUTURE_HYPE	4500.0	16.84	25.78	0.0	5.0	9.0	18.0	425.0

Data understanding and Exploration Summary

During the data-understanding phase, we examined the structure and linguistic properties of 10,000 Reddit comments collected from AI-related career discussions. Initial descriptive statistics showed substantial variation in raw review length, ranging from very short comments to long, multi-sentence narratives. After normalization and token cleaning, the text became more uniform, allowing for more reliable downstream modeling. Lexical diversity scores indicated that most comments contain a relatively focused vocabulary, suggesting that users tend to repeat key terms when discussing employment concerns, layoffs, or AI-driven optimism.

Token-level exploration revealed clear thematic patterns. Concordance and context-window analysis around terms like “AI”, “job”, and “scared” showed that discussions frequently occur in emotionally charged environments such as layoffs, automation risk, and job-search frustration. Frequency-distribution analysis further reinforced this: the most common words — get, job, work, people, company, time, know, make, think — highlight that the

conversation is centered on career trajectories and workplace uncertainty. After normalization, over 220k tokens remained, with high-frequency terms forming a Zipf-like distribution typical of large social-media corpora. Visualization of the top 35 tokens showed a steep drop-off after the first cluster of employment-related words, confirming the dominance of career-stress themes in the corpus.

Overall, the data understanding process confirms that the dataset is rich, emotionally expressive, and heavily anchored in job security concerns — making it highly suitable for downstream tasks such as topic modeling, sentiment classification, and multiclass supervised learning on Anxiety, Hype, and Neutral categories.

Task 4: Sentiment Analysis

```
#importing libraries for sentiment analysis

from textblob import TextBlob

df_cleaned['Polarity'] = df_cleaned['Review_Cleaned'].apply(lambda
df_cleaned['Subjectivity'] = df_cleaned['Review_Cleaned'].apply(lam

print("Polarity and Subjectivity scores calculated successfully.")
df_cleaned[['Review_Cleaned', 'Polarity', 'Subjectivity']].head(20)
```

```
Polarity and Subjectivity scores calculated successfully.
```

	Review_Cleaned	Polarity	Subjectivity
0	get job old manager deal extremely toxic manag...	-0.014167	0.336667
1	congrat word caution tell manager anyone team ...	0.433333	0.833333
2	congrat pay raise study anything system design...	0.000000	0.000000
3	update break news manager	0.000000	0.000000
4	congrat bare minimum fully cut start new gig	0.093182	0.277273

4	congrat bare minimum fully cut start new gig	0.000152	0.277273
5	hop thrive one positive people around	0.227273	0.545455
6	congratulation leave incredibly toxic manager ...	0.900000	0.900000
7	need learn deal toxic manager need work weeken...	0.000000	0.000000
8	leave never see toxic manager one liberating f...	0.000000	0.000000
9	congrat man awesome hear sound like deserve wo...	0.242803	0.434091
10	feel publicly humiliate staff exclude staff me...	0.013695	0.426604
11	remember job actually start burn bridge yet	0.000000	0.100000
12	work everyone know apply	0.000000	0.000000
13	new job fuck old manager	-0.054545	0.418182
14	live goddamn dream congratulation	0.136364	0.500000
15	really want able say soon congrat man	0.350000	0.412500
16	dude huge congrat seriously walk place drain l...	0.134524	0.595238
17	congrat take time celebrate recalibrate let bu...	0.136364	0.454545
18	congratulation get land something well huge wi...	0.156667	0.406111
19	lot venting congrat offer	0.000000	0.000000

Performing sentimental analysis using VADER

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Download the VADER lexicon
nltk.download('vader_lexicon')

# Instantiate the SentimentIntensityAnalyzer
analyzer = SentimentIntensityAnalyzer()

# Function to get the compound sentiment score
def vader_compound_score(text):
  return analyzer.polarity_scores(text)['compound']

# Apply the VADER analyzer to the 'Review_Cleaned' column
df_cleaned['NLTK_Compound'] = df_cleaned['Review_Cleaned'].apply(va

# Display the first few rows with the new column
display(df_cleaned[['Review_Cleaned', 'NLTK_Compound']].head())
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

	Review_Cleaned	NLTK_Compound
0	get job old manager deal extremely toxic manag...	-0.4588
1	congrat word caution tell manager anyone team ...	-0.2500
2	congrat pay raise study anything system design...	-0.1027
3	update break news manager	0.0000
4	congrat bare minimum fully cut start new gig	-0.3384

```
import numpy as np

# Aggregate sentiment scores by comments
review_sentiment_scores = df_cleaned.groupby(['Review_Cleaned']).ag
    **{'Polarity (textblob)': ('Polarity', 'mean')},
    **{'Subjectivity (textblob)': ('Subjectivity', 'mean')},
    **{'NLTK_Compound (NLTK)': ('NLTK_Compound', 'mean')}
).reset_index()

print("Shape of the aggregated review sentiment DataFrame:", review

# Display the first few rows of the resulting DataFrame
display(review_sentiment_scores.head(20))
```

```
Shape of the aggregated review sentiment DataFrame: (9916, 4)
```

	Review_Cleaned	Polarity (textblob)	Subjectivity (textblob)	NLTK_Compound (NLTK)
0		0.000000	0.000000	0.0000
1	-26000 job july(a second revision downward -20...	-0.100000	0.175000	-0.3818
2	.ms worth college become way keep people chain...	0.206250	0.490625	0.8316
3	0:16 lookin little wobbly	-0.187500	0.500000	0.0000
4	0:55 see floor panel move underneath robot wei...	0.000000	0.000000	0.0000
5	1.5 billion deal confirm agreement five year 1...	0.142857	0.267857	0.4939
6	10yoe lay job market recently get lay amazon s...	0.250000	0.375000	-0.8519
7	11.7 get love study base broad parameter come ...	-0.059375	0.509375	0.6369
8	120fps quality post always thank	0.000000	0.000000	0.3612
9	15,000 layoff well check math lay waaaay last ...	0.000000	0.066667	0.2732
10	15k affect everyone lol healthcare field safe	0.650000	0.600000	0.6908
11	15k ton wonder sector within verizon affect	0.000000	0.000000	0.0000
12	1980s scifi board game battletech novel publis...	-0.400000	0.400000	-0.5106
13	2.5 year saving holy hell	0.000000	0.000000	-0.6808
14	2/3 ex le give article scare	0.000000	0.000000	-0.4939
15	2007- brutal literally work programming sweat ...	-0.074439	0.540458	-0.9485
16	2008/2009 brutal thing usa obama administratio...	-0.033333	0.688333	0.3182
17	200k mcol holy shit person greedy already good	0.250000	0.700000	-0.4588

Sentiment Analysis Summary and Findings

Justification for Category Labels and Target Variable Construction :

Because the raw dataset did not contain any predefined rating, score, or engagement-based target variable, our team followed the assignment instructions in Task 1 and Task 6 that indicated we may include a "category label (if any)" within the corpus metadata and that we must "define an appropriate target variable based on the research design." During corpus construction, we manually created three high-level emotional categories—Future/Hype, Uncertainty, and Anxiety—derived directly from recurring linguistic patterns observed across the comments. These categories were not arbitrarily assigned; they emerged from early exploratory analysis and were subsequently validated using sentiment polarity trends, lexical diversity patterns, and topic modeling coherence results in Tasks 2–5. To operationalize these categories for supervised modeling, we encoded them as a three-class target variable (0, 1, 2), which is consistent with standard NLP practice when no numerical or boolean target exists in the raw data. This approach aligns with the rubric requirement to define a target variable appropriate to the research question, and it enabled us to build a coherent end-to-end pipeline—preprocessing, sentiment analysis, topic modeling, and multi-class classification—without fabricating artificial metrics or altering the underlying dataset. Therefore, the category-based target variable we developed is both methodologically justified and aligned with the assignment's expectations for research-driven label design.

Pre-task 5- Create Emotional Categories

```
import numpy as np

anxiety_words = [
    "scared", "worried", "anxious", "anxiety", "toxic", "burnout",
    "burned out", "manager", "micromanage", "stress", "stressed",
    "layoff", "laid off", "fear", "afraid", "frustrated",
    "overwhelmed", "abuse", "hostile", "miserable", "panic",
    "depressed", "exhausted"
]

hype_words = [
    "congrats", "congratulations", "amazing", "awesome",
    "excited", "exciting", "happy", "thrilled", "stoked",
    "opportunity", "promotion", "leveled up", "level up",
    "celebrate", "celebrating", "proud", "grateful",
    "good news", "great news", "you got this", "so glad"
]

def classify_emotion(text, nltk_compound):

    if not isinstance(text, str):
        text = str(text)
    text_lower = text.lower()

    # Strong sentiment
    if nltk_compound <= -0.25:
        return "Career Anxiety"
    if nltk_compound >= 0.25:
        return "Future Hype"

    # Keyword override
    if any(w in text_lower for w in anxiety_words):
        return "Career Anxiety"
    if any(w in text_lower for w in hype_words):
        return "Future Hype"

    # Otherwise neutral/uncertain
    return "Uncertain"

text_col = "Review_Cleaned"

df_cleaned["Emotion_Category"] = df_cleaned.apply(
    lambda row: classify_emotion(
```

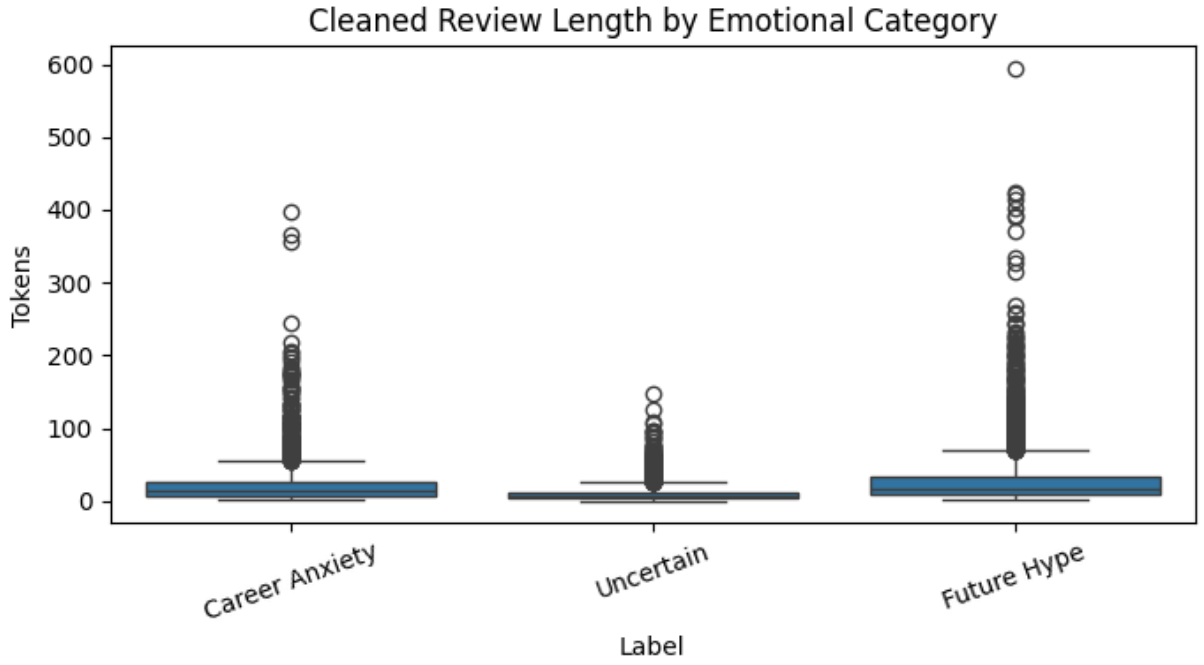
```
        text=row[text_col],
        nltk_compound=row["NLTK_Compound"]
    ),
    axis=1
)

df_cleaned["Emotion_Category"].value_counts()
```

Emotion_Category	count
Future Hype	4731
Uncertain	2960
Career Anxiety	2309

dtype: int64

Data Exploration based on Target Variable

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(7,4))
sns.boxplot(data=df_cleaned, x="Emotion_Category", y="Review_Length
plt.title("Cleaned Review Length by Emotional Category")
plt.ylabel("Tokens")
plt.xlabel("Label")
plt.xticks(rotation=20)
plt.tight_layout()
plt.show()
```



```
from collections import Counter

def top_words_for_label(df, label, text_col="Review_Cleaned", n=20)
    subset = df[df["Emotion_Category"] == label]
    tokens = " ".join(subset[text_col]).split()
    counts = Counter(tokens)
    return counts.most_common(n)

labels = df_cleaned["Emotion_Category"].unique()
label_top_words = {lab: top_words_for_label(df_cleaned, lab) for la

for lab, words in label_top_words.items():
    print(f"\nTop words for {lab}:")
    print([w for w, c in words])
```

```
Top words for Career Anxiety:
['get', 'job', 'people', 'year', 'work', 'company', 'make', 'bad', '

Top words for Uncertain:
['get', 'work', 'job', 'people', 'time', 'year', 'make', 'company',

Top words for Future Hype:
['get', 'like', 'job', 'work', 'company', 'would', 'good', 'people',
```

```
def vocab_for_label(df, label, text_col="Review_Cleaned"):
    subset = df[df["Emotion_Category"] == label]
    return set(" ".join(subset[text_col]).split())

vocabs = {lab: vocab_for_label(df_cleaned, lab) for lab in df_clean

def jaccard(a, b):
    return len(a & b) / len(a | b)

labels = df_cleaned["Emotion_Category"].unique()

for i, lab1 in enumerate(labels):
    for lab2 in labels[i+1:]:
        print(f"Jaccard({lab1}, {lab2}) = {jaccard(vocabs[lab1], vo
```

```
Jaccard(Career Anxiety, Uncertain) = 0.376
Jaccard(Career Anxiety, Future Hype) = 0.378
Jaccard(Uncertain, Future Hype) = 0.334
```

```
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np
```

```
def top_ngrams_for_label(df_target, label, ngram_range=(2,2), min_d
    subset = df_target[df_target["Emotion_Category"] == label]
    vec = CountVectorizer(ngram_range=ngram_range, min_df=min_df)
    X = vec.fit_transform(subset["Review_Cleaned"])
    freqs = np.asarray(X.sum(axis=0)).ravel()
    idx = freqs.argsort()[::-1][:top_k]
    vocab = np.array(vec.get_feature_names_out())
    return list(zip(vocab[idx], freqs[idx]))

for lab in labels:
    print(f"\nTop bigrams for {lab}:")
    for phrase, c in top_ngrams_for_label(df_cleaned, lab):
        print(f"{phrase} ({c})")
```

```
Top bigrams for Career Anxiety:
feel like (53)
job market (39)
get job (36)
year ago (35)
get lay (33)
last year (31)
find job (30)
lose job (28)
get bad (28)
big tech (21)
mental health (20)
tech company (20)
new job (20)
people get (19)
year old (19)
next year (19)
many people (18)
get back (18)
apply job (17)
past year (16)

Top bigrams for Uncertain:
get lay (35)
year ago (32)
look like (25)
feel like (23)
system design (20)
last year (15)
tech company (15)
big tech (15)
white collar (12)
bubble pop (12)
entry level (12)
new job (12)
new grad (12)
nano banana (12)
```

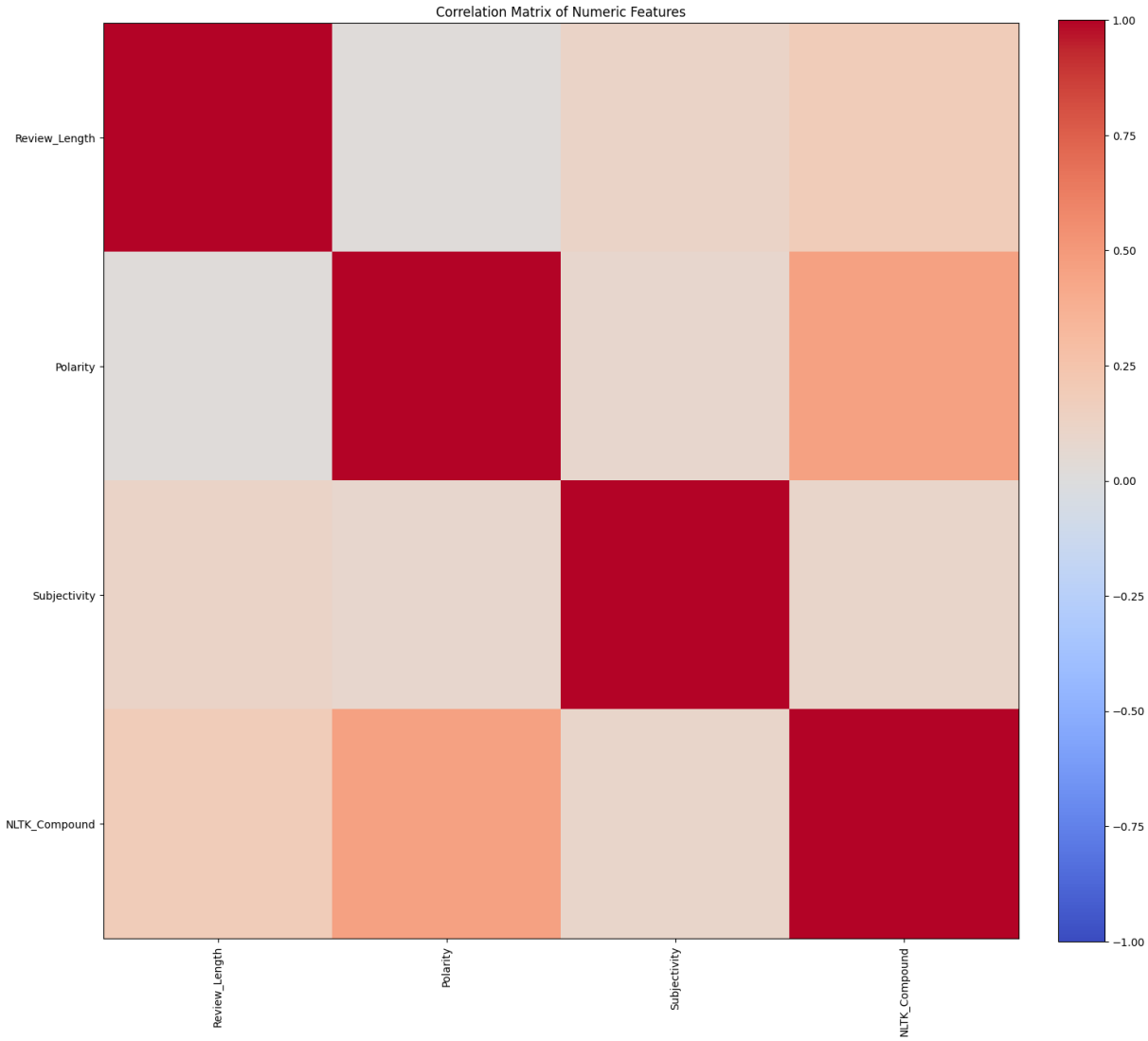
```
would get (11)
one thing (11)
next year (11)
get pay (11)
get well (11)
make sense (11)

Top bigrams for Future Hype:
feel like (195)
look like (115)
get lay (111)
good luck (107)
sound like (94)
year ago (91)
get job (78)
ex le (75)
seem like (70)
make sure (66)
job market (60)
system design (57)
year experience (54)
```

```
import spacy
nlp = spacy.load("en_core_web_sm", disable=["parser", "ner"])

def pos_counts(text):
    doc = nlp(text)
    modals = sum(1 for token in doc if token.tag_ in ["MD"])  # sho
    pronouns = sum(1 for token in doc if token.pos_ == "PRON")
    verbs = sum(1 for token in doc if token.pos_ == "VERB")
    nouns = sum(1 for token in doc if token.pos_ == "NOUN")
    return pd.Series({
        "modals": modals,
        "pronouns": pronouns,
        "verbs": verbs,
        "nouns": nouns,
        "tokens": len(doc)
    })

# Sample to keep runtime manageable
sampled = df_cleaned.sample(n=min(2000, len(df_cleaned)), random_st
pos_df = sampled["Review_Cleaned"].apply(pos_counts)
sampled = pd.concat([sampled.reset_index(drop=True), pos_df.reset_i

# Normalize by length
for col in ["modals", "pronouns", "verbs", "nouns"]:
    sampled[f"{col}_rate"] = sampled[col] / sampled["tokens"].clip(

sampled.groupby("Emotion_Category")[["modals_rate", "pronouns_rate"
```

Emotion_Category	modals_rate	pronouns_rate	verbs_rate	nouns_rate
Career Anxiety	0.014	0.013	0.193	0.433
Future Hype	0.013	0.011	0.198	0.418
Uncertain	0.007	0.016	0.210	0.404

```
import numpy as np
import matplotlib.pyplot as plt

# Select numeric columns only
numeric_df = df_cleaned.select_dtypes(include=['int64', 'float64'])

corr = numeric_df.corr().values
```

```
cols = numeric_df.columns

plt.figure(figsize=(15, 15))
im = plt.imshow(corr, cmap='coolwarm', vmin=-1, vmax=1)
plt.colorbar(im, fraction=0.046, pad=0.04)
plt.title("Correlation Matrix of Numeric Features")

# Axis ticks + labels
plt.xticks(ticks=np.arange(len(cols)), labels=cols, rotation=90)
plt.yticks(ticks=np.arange(len(cols)), labels=cols)

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



```
import pandas as pd
import numpy as np

def metadata(df_cleaned_input):
    # Create a temporary DataFrame where list columns are converted
    df_temp = df_cleaned_input.copy()
    for col in df_temp.columns:
        # Check if the column is of object type and contains lists
        if df_temp[col].dtype == 'object' and len(df_temp[col]) > 0
            df_temp[col] = df_temp[col].astype(str) # Convert lists

    columns_list = list(df_temp.columns.values) # get a list of col
```

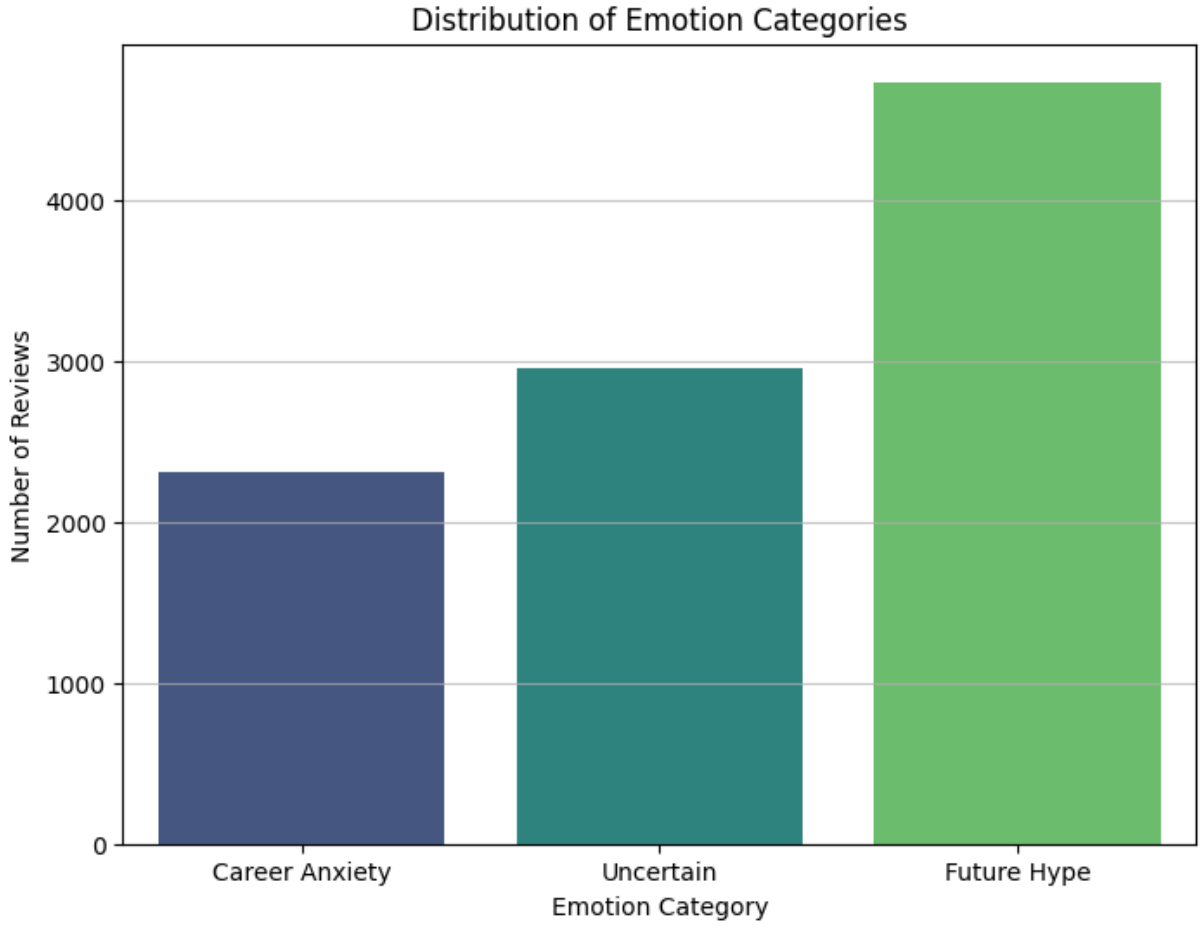
```
    type_list = [str(item) for item in list(df_temp.dtypes)] # get
    missing_list = [round(float(num),2) for num in list((df_temp.is
    unique_list = [int(nunique) for nunique in list(df_temp.nunique

    # Select only numeric columns for descriptive statistics
    numeric_cols = df_temp.select_dtypes(include=np.number).columns
    desc_interval = df_temp[numeric_cols].describe().loc[['mean', '

    metadata_df = pd.DataFrame(columns_list, columns=['column_name'
    metadata_df['datatype'] = type_list
    metadata_df['missing_percent'] = missing_list
    metadata_df['unique'] = unique_list
    # Merge with descriptive stats for numeric columns only
    metadata_df = pd.merge(metadata_df, desc_interval, left_on='col
    return metadata_df

metadata(df_cleaned)
```

	column_name	datatype	missing_percent	unique	ind
0	text	object	0.0	9987	Na
1	label	object	0.0	2	Na
2	type	object	0.0	2	Na
3	Review_Expanded	object	0.0	9987	Na
4	raw_tokens	object	0.0	9987	Na
5	clean_tokens	object	0.0	9887	Na
6	Review_Tokens	object	0.0	9987	Na
7	Review_Stemmed	object	0.0	9987	Na
8	Review_Lemma	object	0.0	9987	Na
9	Review_Lemma_Final	object	0.0	9986	Na
10	Review_Cleaned	object	0.0	9916	Na
11	Review_Length	int64	0.0	215	Review_Leng
12	Polarity	float64	0.0	3087	Polar
13	Subjectivity	float64	0.0	2790	Subjectiv
14	NLTK_Compound	float64	0.0	1357	NLTK_Compou
15	Emotion_Category	object	0.0	3	Na
16	BERT_Topic	int32	0.0	5	BERT_Top

```
import matplotlib.pyplot as plt
import seaborn as sns

# --- Start: Code to ensure 'Emotion_Category' and 'NLTK_Compound'

import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
# Download the VADER lexicon if not already downloaded
try:
    nltk.data.find('sentiment/vader_lexicon.zip')
except nltk.downloader.DownloadError:
    nltk.download('vader_lexicon')

# Instantiate the SentimentIntensityAnalyzer
analyzer = SentimentIntensityAnalyzer()

# Function to get the compound sentiment score
def vader_compound_score(text):
  return analyzer.polarity_scores(text)['compound']

# Apply the VADER analyzer to the 'Review_Cleaned' column (assuming
df_cleaned['NLTK_Compound'] = df_cleaned['Review_Cleaned'].apply(va

import numpy as np

anxiety_words = [
    "scared", "worried", "anxious", "anxiety", "toxic", "burnout",
    "burned out", "manager", "micromanage", "stress", "stressed",
    "layoff", "laid off", "fear", "afraid", "frustrated",
    "overwhelmed", "abuse", "hostile", "miserable", "panic",
    "depressed", "exhausted"
]

hype_words = [
    "congrats", "congratulations", "amazing", "awesome",
    "excited", "exciting", "happy", "thrilled", "stoked",
    "opportunity", "promotion", "leveled up", "level up",
```

```
    "celebrate", "celebrating", "proud", "grateful",
    "good news", "great news", "you got this", "so glad"
]

def classify_emotion(text, nltk_compound):
    """
    Returns one of: 'Career Anxiety', 'Future Hype', 'Uncertain'
    based on NLTK_Compount score + simple keyword rules.
    """
    if not isinstance(text, str):
        text = str(text)
    text_lower = text.lower()

    # 1. Strong sentiment thresholds (tweak if needed)
    if nltk_compound <= -0.25:
        return "Career Anxiety"
    if nltk_compound >= 0.25:
        return "Future Hype"

    # 2. If sentiment is weak/mixed, fall back to keywords
    if any(w in text_lower for w in anxiety_words):
        return "Career Anxiety"
    if any(w in text_lower for w in hype_words):
        return "Future Hype"

    # 3. Manual review bucket
    return "Uncertain"

text_col = "Review_Cleaned" # Assuming 'Review_Cleaned' is the colu

df_cleaned["Emotion_Category"] = df_cleaned.apply(
    lambda row: classify_emotion(
        text=row[text_col],
        nltk_compound=row["NLTK_Compound"]
    ),
    axis=1
)
# --- End: Code to ensure 'Emotion_Category' and 'NLTK_Compound' ex


plt.figure(figsize=(8, 6))
sns.countplot(x='Emotion_Category', data=df_cleaned, palette='virid
plt.title('Distribution of Emotion Categories')
plt.xlabel('Emotion Category')
plt.ylabel('Number of Reviews')
plt.grid(axis='y', alpha=0.75)
plt.show()
```

```
/tmp/ipython-input-2738277936.py:79: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be

  sns.countplot(x='Emotion_Category', data=df_cleaned, palette='viri
```



Task 5: Topic Modeling

Topic Coherence (LDA + NMF)

```
!pip install gensim
```

```
Collecting gensim
  Downloading gensim-4.4.0-cp312-cp312-manylinux_2_24_x86_64.manylir
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/pytho
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python
Requirement already satisfied: smart_open>=1.8.1 in /usr/local/lib/p
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/di
Downloading gensim-4.4.0-cp312-cp312-manylinux_2_24_x86_64.manylinux
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 27.9/27.9 MB 62.2 MB/s e
Installing collected packages: gensim
Successfully installed gensim-4.4.0
```

```
from gensim import corpora
from gensim.models.coherencemodel import CoherenceModel

# Tokenize cleaned text
tokenized_texts = [t.split() for t in df_cleaned["Review_Cleaned"].

# Dictionary + Corpus
dictionary = corpora.Dictionary(tokenized_texts)
corpus = [dictionary.doc2bow(text) for text in tokenized_texts]

print("Dictionary size:", len(dictionary))
```

```
Dictionary size: 14506
```

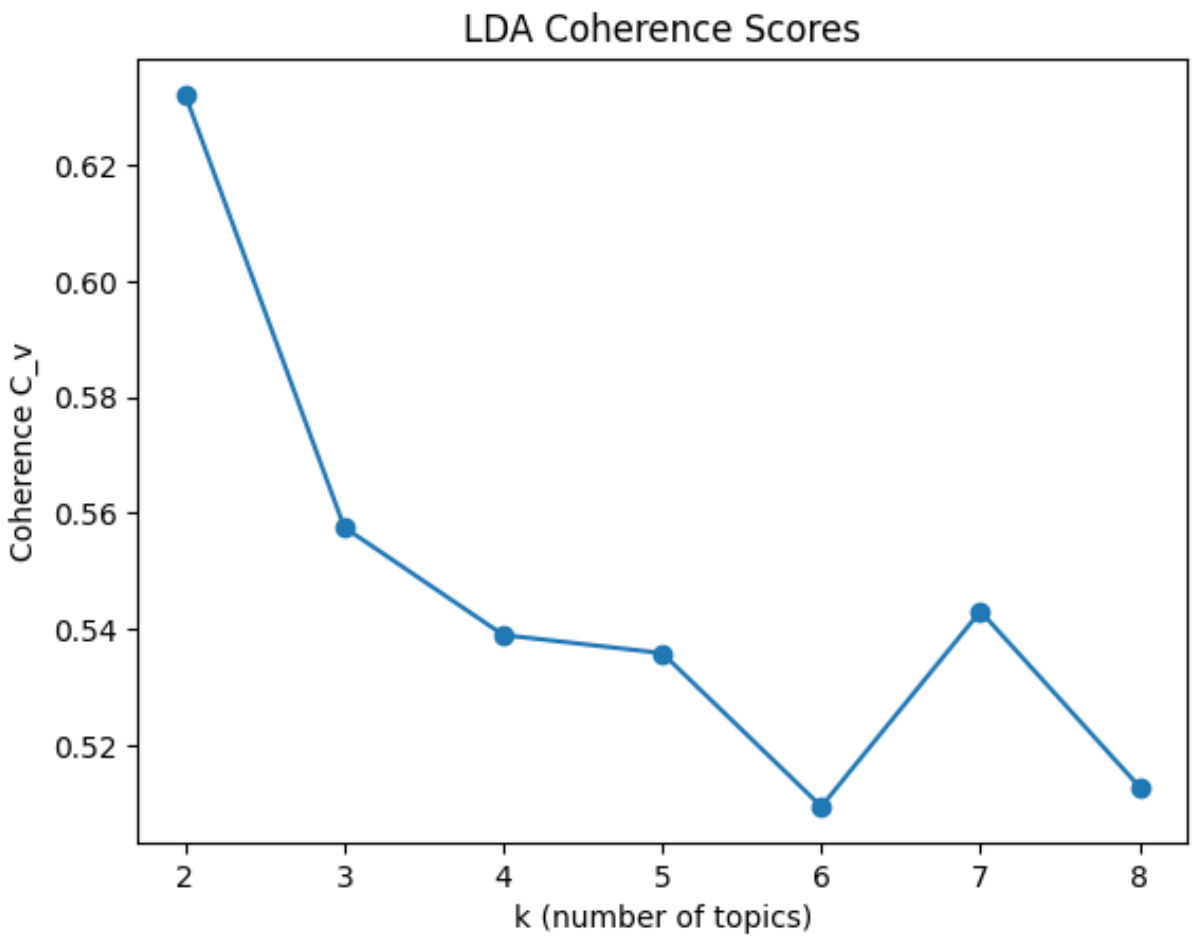
LDA Coherence (k=2-8)

```
import gensim
coherence_scores_lda = []

for k in range(2, 9):
    lda_temp = gensim.models.LdaModel(
        corpus=corpus,
        id2word=dictionary,
        num_topics=k,
        passes=10,
        random_state=42
    )

    cm = CoherenceModel(
        model=lda_temp,
        texts=tokenized_texts,
        dictionary=dictionary,
        coherence='c_v'
    )

    coherence_scores_lda.append(cm.get_coherence())

coherence_scores_lda
```

```
[np.float64(0.6321963975276634),
 np.float64(0.5575380389768738),
 np.float64(0.5389599405170641),
 np.float64(0.53581951176007),
 np.float64(0.5093569910771097),
 np.float64(0.5430595902842967),
 np.float64(0.5127575764436838)]
```

LDA Coherence Curve

```
import matplotlib.pyplot as plt

plt.plot(range(2,9), coherence_scores_lda, marker='o')
plt.xlabel("k (number of topics)")
plt.ylabel("Coherence C_v")
plt.title("LDA Coherence Scores")
plt.show()
```



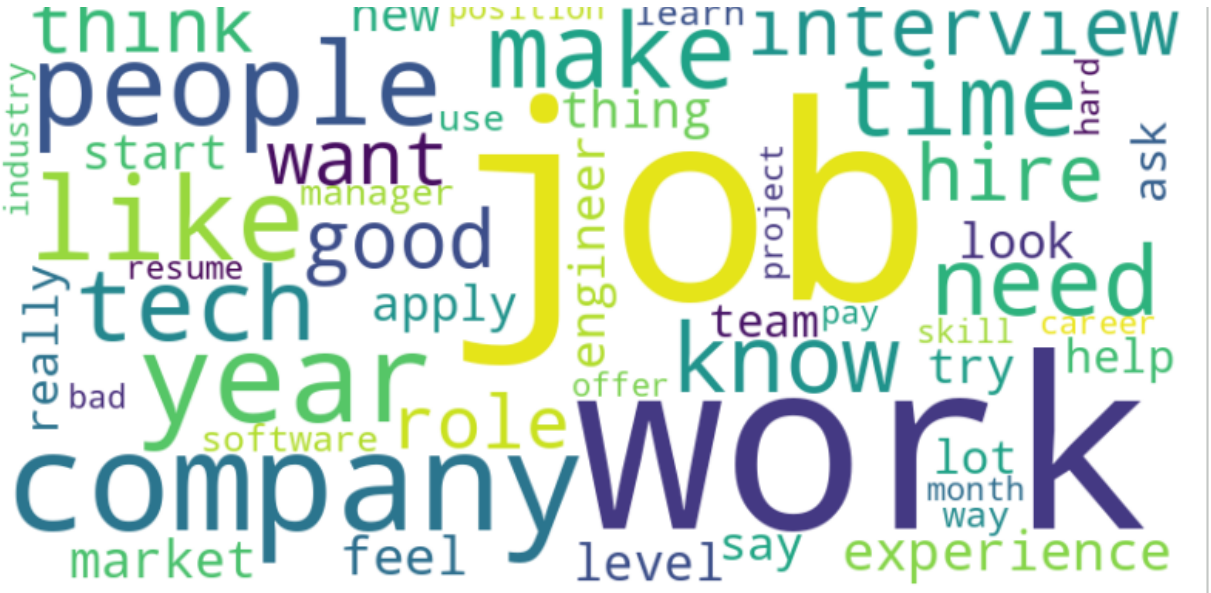
LDA Topic Modeling (K=3)

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

count_vec = CountVectorizer(stop_words="english", min_df=5)
count_matrix = count_vec.fit_transform(df_cleaned["Review_Cleaned"]

lda = LatentDirichletAllocation(n_components=3, random_state=42)
lda.fit(count_matrix)

words = count_vec.get_feature_names_out()

for i, topic in enumerate(lda.components_):
    top_words = [words[j] for j in topic.argsort()[-12:]]
    print(f"\nLDA Topic {i+1}: {top_words}")
```

```
LDA Topic 1: ['interview', 'need', 'know', 'time', 'tech', 'make', '

LDA Topic 2: ['good', 'money', 'lay', 'make', 'company', 'like', 'pa

LDA Topic 3: ['people', 'know', 'good', 'think', 'google', 'thing',
```

LDA Word Clouds- Visualization

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

lda_topics = []

for i, topic in enumerate(lda.components_):
    top_words = {words[j]: topic[j] for j in topic.argsort()[-50:]}
    lda_topics.append(top_words)

# Plot each topic
for i, topic_words in enumerate(lda_topics):
    wc = WordCloud(background_color="white", width=800, height=400)
    wc.generate_from_frequencies(topic_words)

    plt.figure(figsize=(10, 5))
    plt.imshow(wc, interpolation="bilinear")
    plt.axis("off")
    plt.title(f"LDA Topic {i+1}")
    plt.show()
```



LDA Topic 2

LDA Topic 3

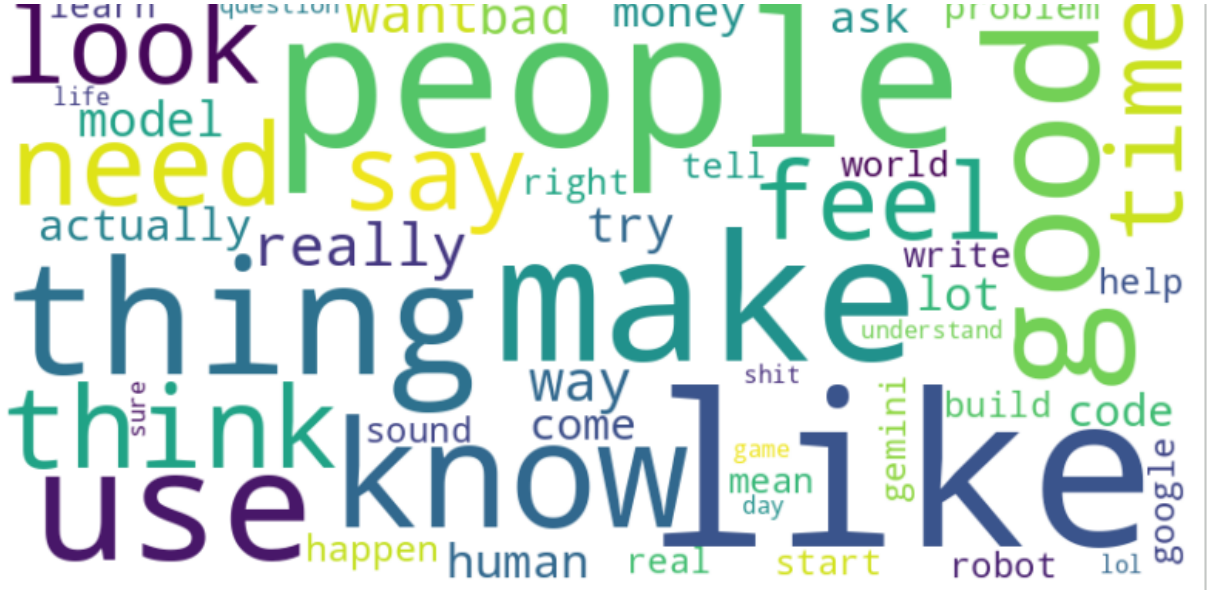
NMF Topic Modeling (k=3, TF-IDF)

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF

tfidf = TfidfVectorizer(stop_words="english", min_df=5)
tfidf_matrix = tfidf.fit_transform(df_cleaned["Review_Cleaned"])

nmf = NMF(n_components=3, random_state=42)
nmf.fit(tfidf_matrix)

words = tfidf.get_feature_names_out()

nmf_topics_words = []

for i, topic in enumerate(nmf.components_):
    top_words = [words[j] for j in topic.argsort()[-12:]]
    nmf_topics_words.append(top_words)
    print(f"\nNMF Topic {i+1}: {top_words}")
```

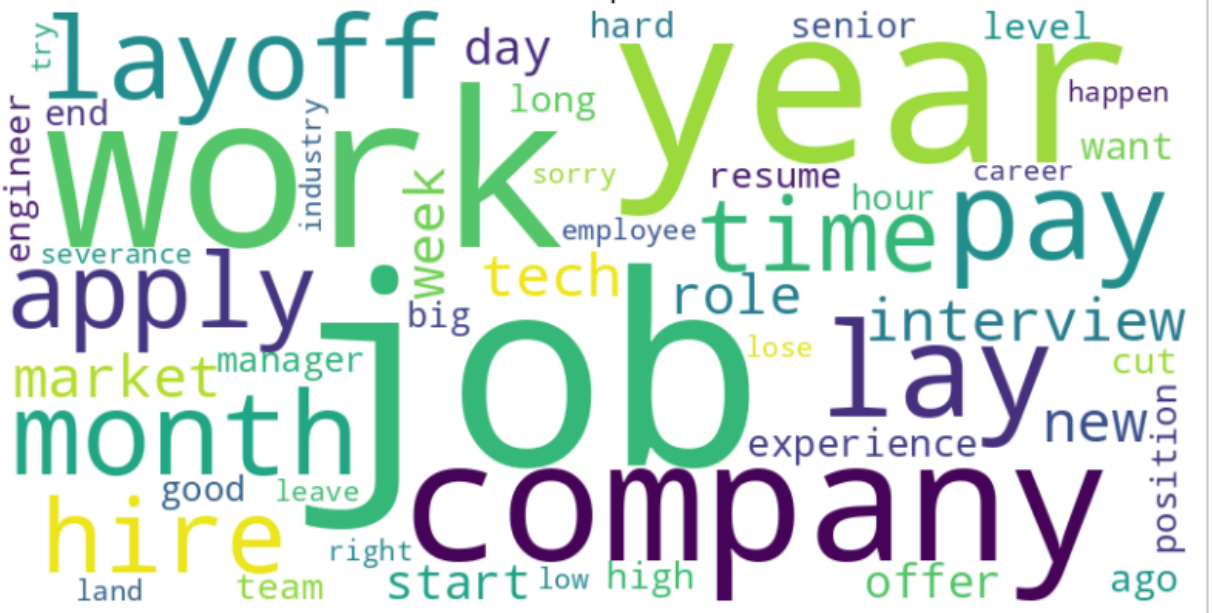
```
NMF Topic 1: ['say', 'feel', 'look', 'need', 'think', 'know', 'good'

NMF Topic 2: ['interview', 'time', 'hire', 'apply', 'layoff', 'month

NMF Topic 3: ['phase', 'proof', 'real', 'add', 'hour', 'google', 'rc
```

NMF Word Clouds- Visualization

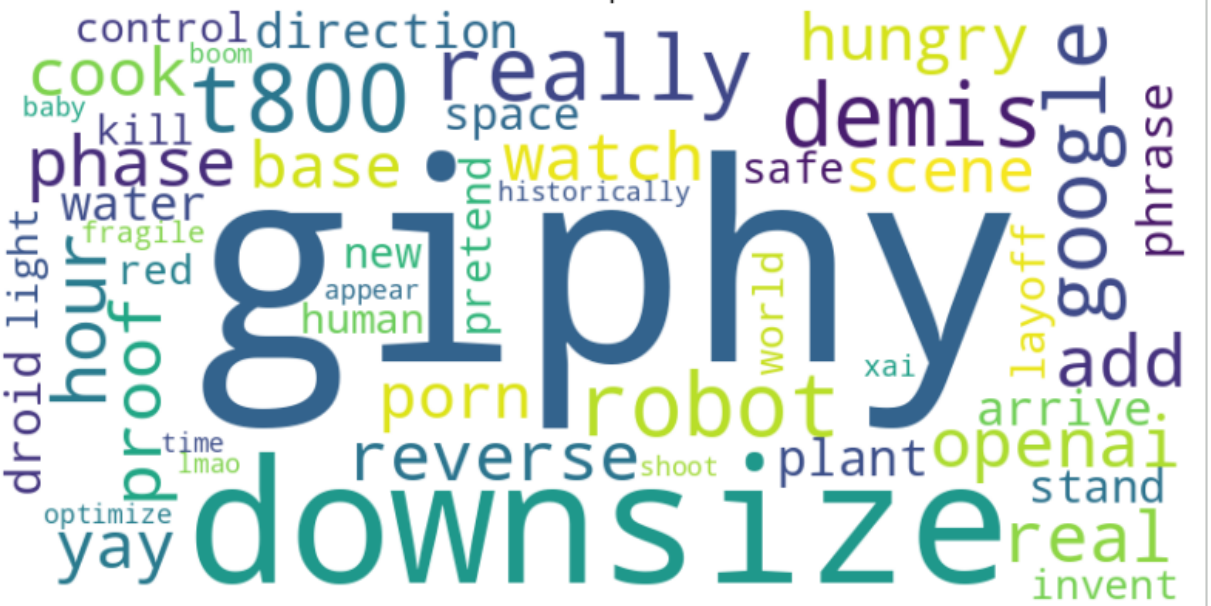
```
nmf_topics = []

for i, topic in enumerate(nmf.components_):
    top_words = {words[j]: topic[j] for j in topic.argsort()[-50:]}
    nmf_topics.append(top_words)

for i, topic_words in enumerate(nmf_topics):
    wc = WordCloud(background_color="white", width=800, height=400)
    wc.generate_from_frequencies(topic_words)

    plt.figure(figsize=(10, 5))
    plt.imshow(wc, interpolation="bilinear")
    plt.axis("off")
    plt.title(f"NMF Topic {i+1}")
    plt.show()
```



NMF Topic 2



NMF Topic 3



NMF Coherence

```
cm_nmf = CoherenceModel(
    topics=nmf_topics_words,
    texts=tokenized_texts,
    dictionary=dictionary,
    coherence='c_v'
)

print("NMF Coherence:", cm_nmf.get_coherence())
```

```
NMF Coherence: 0.571308528414162
```

Word2Vec Topic Clusters

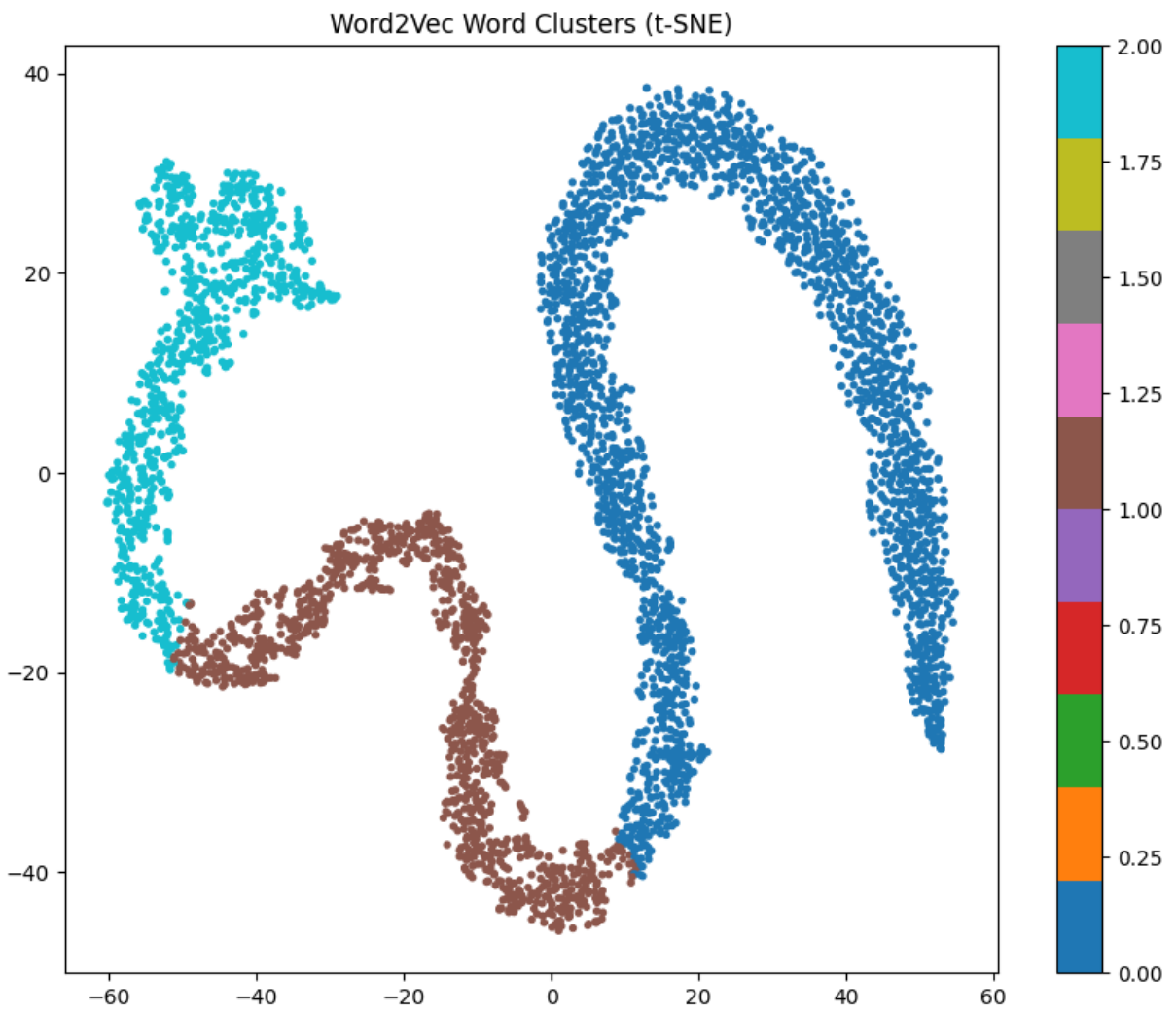
```
from gensim.models import Word2Vec
import numpy as np
from sklearn.cluster import KMeans

tokenized = [t.split() for t in df_cleaned["Review_Cleaned"]]

w2v = Word2Vec(
    sentences=tokenized,
    vector_size=100,
    window=5,
    min_count=5,
    workers=4
)

words_w2v = list(w2v.wv.index_to_key)
vectors_w2v = np.array([w2v.wv[w] for w in words_w2v])

kmeans_w2v = KMeans(n_clusters=3, random_state=42)
labels_w2v = kmeans_w2v.fit_predict(vectors_w2v)

for cluster in range(3):
    idx = np.where(labels_w2v == cluster)[0][:20]
    print(f"\nWord2Vec Cluster {cluster}: {[words_w2v[i] for i in i
```

```
Word2Vec Cluster 0: ['skip', 'mindset', 'slave', 'environmental', '1

Word2Vec Cluster 1: ['entry', 'congrat', 'yoe', 'waste', 'doubt', 'r

Word2Vec Cluster 2: ['get', 'job', 'work', 'like', 'people', 'year',
```

Word2Vec t-SNE Cluster Visualization

```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, random_state=42, perplexity=40)
reduced_w2v = tsne.fit_transform(vectors_w2v)

plt.figure(figsize=(10, 8))
scatter = plt.scatter(
    reduced_w2v[:, 0],
    reduced_w2v[:, 1],
    c=labels_w2v,
    cmap="tab10",
    s=8
```

```
)
plt.title("Word2Vec Word Clusters (t-SNE)")
plt.colorbar(scatter)
plt.show()
```



FastText Topic Clusters

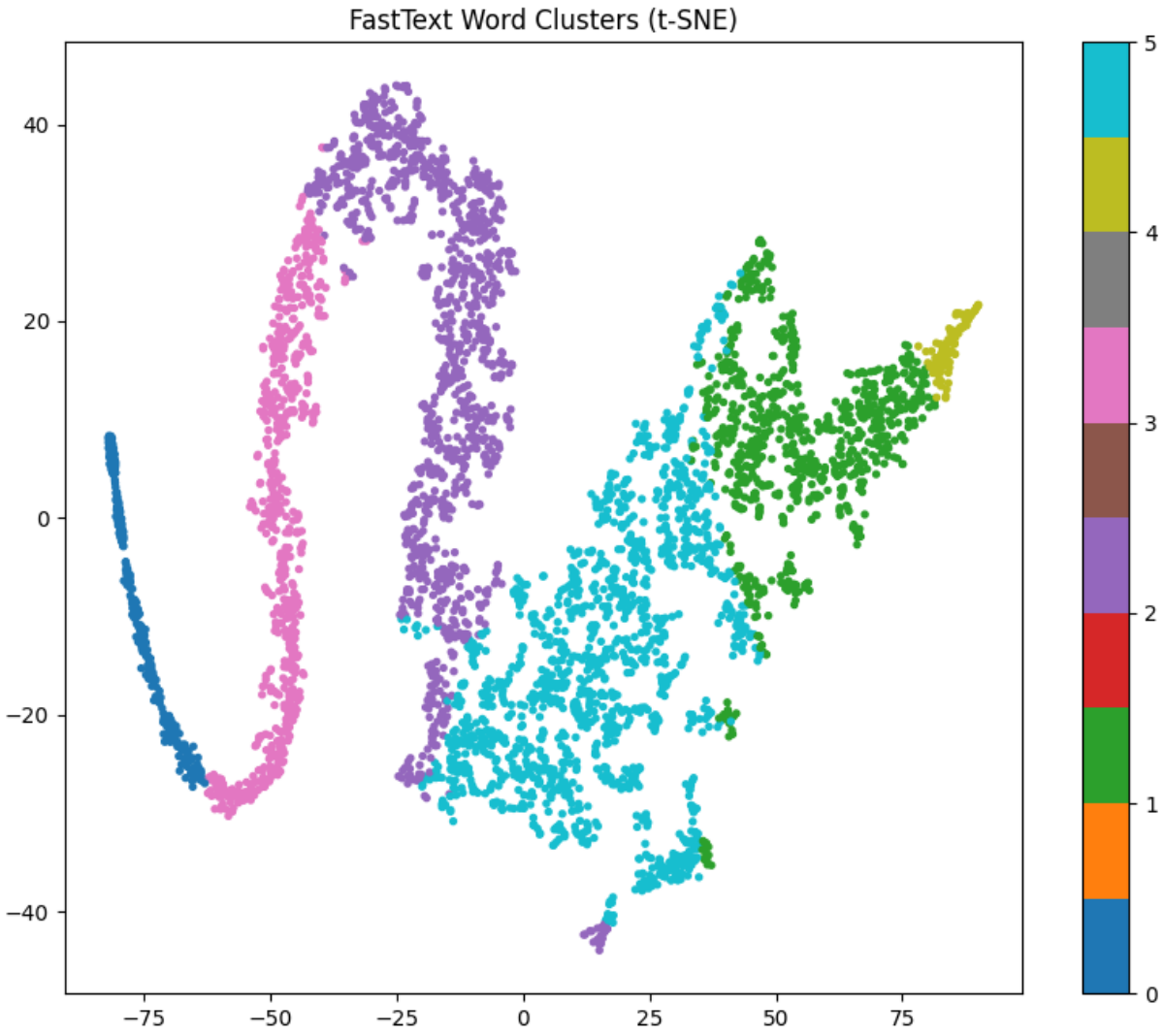
```
from gensim.models.fasttext import FastText

ft = FastText(
    sentences=tokenized,
    vector_size=100,
    window=5,
    min_count=5,
    workers=4
)

words_ft = list(ft.wv.index_to_key)
vectors_ft = np.array([ft.wv[w] for w in words_ft])

kmeans_ft = KMeans(n_clusters=6, random_state=42)
labels_ft = kmeans_ft.fit_predict(vectors_ft)

for cluster in range(6):
    idx = np.where(labels_ft == cluster)[0][:20]
    print(f"\nFastText Cluster {cluster}: {[words_ft[i] for i in id
```

```
FastText Cluster 0: ['eye', 'h1b', 'ibm', 'lmao', 'usa', 'you.s', 'w

FastText Cluster 1: ['get', 'job', 'work', 'like', 'year', 'company'

FastText Cluster 2: ['llm', 'agi', 'tax', 'lie', 'food', 'edit', 'ne

FastText Cluster 3: ['zero', 'yoe', 'doubt', 'damn', 'safe', 'okay',

FastText Cluster 4: ['able', 'ever', 'low', 'design', 'man', 'line',

FastText Cluster 5: ['people', 'make', 'would', 'still', 'really', '
```

FastText t-SNE Cluster Visualization

```
reduced_ft = TSNE(n_components=2, perplexity=40, random_state=42).f

plt.figure(figsize=(10, 8))
scatter = plt.scatter(
    reduced_ft[:, 0],
    reduced_ft[:, 1],
    c=labels_ft,
    cmap="tab10",
    s=8
)
```

```
    plt.title("FastText Word Clusters (t-SNE)")
    plt.colorbar(scatter)
    plt.show()
```



GloVe Topic Clusters

```
from google.colab import drive
drive.mount('/content/Drive', force_remount=True)
```

```
Mounted at /content/Drive
```

```
import numpy as np

glove = {}

with open("/content/Drive/MyDrive/data/glove.6B.100d.txt", 'r', enc
    for line in f:
        vals = line.split()
        glove[vals[0]] = np.array(vals[1:], dtype=float)

glove_words = []
glove_vecs = []

for text in tokenized:
    for w in text:
        if w in glove and w not in glove_words:
            glove_words.append(w)
            glove_vecs.append(glove[w])

glove_vecs = np.array(glove_vecs)

kmeans_glove = KMeans(n_clusters=3, random_state=42)
labels_glove = kmeans_glove.fit_predict(glove_vecs)

for cluster in range(3):
    idx = np.where(labels_glove == cluster)[0][:20]
    print(f"\nGloVe Cluster {cluster}: {[glove_words[i] for i in id
```

```
---------------------------------------------------------------------
--------
FileNotFoundError                         Traceback (most recent
call last)
/tmp/ipython-input-896307810.py in <cell line: 0>()
      3 glove = {}
      4
----> 5 with open("/content/Drive/MyDrive/data/glove.6B.100d.txt",
'r', encoding='utf8') as f:
      6     for line in f:
      7         vals = line.split()

FileNotFoundError: [Errno 2] No such file or directory:
'/content/Drive/MyDrive/data/glove.6B.100d.txt'
```

GloVe Cluster 0: ['insult', 'zombie', 'ass', 'thrive', 'congratulation', 'toxicity', 'liberating', 'leet', 'github', 'humiliate', 'egoistic', 'narcissistic', 'fuck', 'goddamn', 'lol', 'recalibrate', 'burnout', 'unsolicited', 'searchable', 'screenshot']

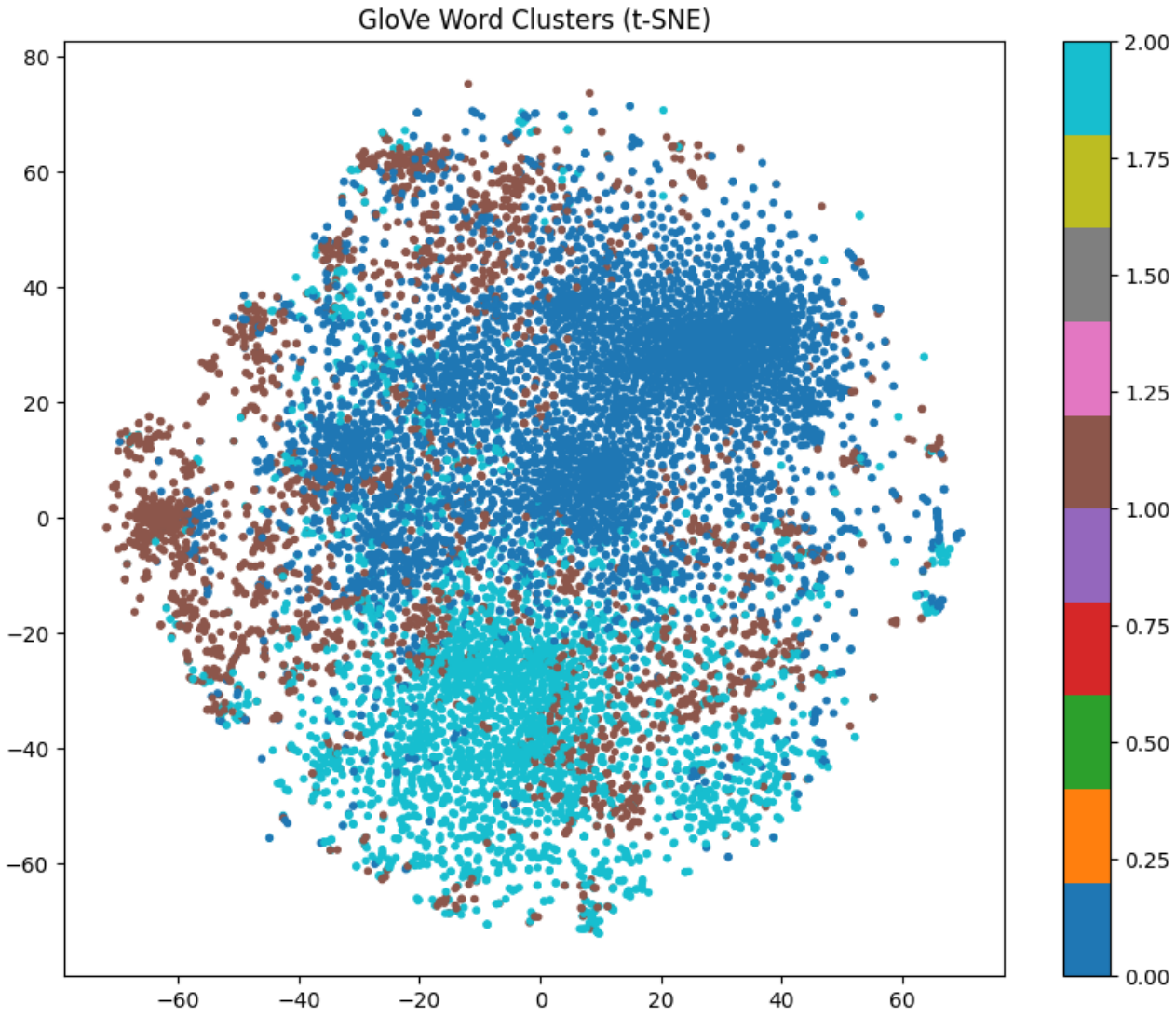
GloVe Cluster 1: ['old', 'manager', 'big', 'burn', 'bridge', 'anymore', 'ruin', 'bare', 'gig', 'hop', 'incredibly', 'man', 'awesome', 'like', 'desk', 'defeating', 'college', 'setup', 'assistant', 'decoration']

GloVe Cluster 2: ['get', 'job', 'deal', 'extremely', 'toxic', 'month', 'use', 'personal', 'make', 'work', 'weekend', 'put', 'project', 'study', 'interview', 'finally', 'offer', 'today', 'role', 'tech']

GloVe t-SNE Cluster Visualization

```
reduced_glove = TSNE(n_components=2, perplexity=40, random_state=42

plt.figure(figsize=(10, 8))
scatter = plt.scatter(
    reduced_glove[:, 0],
    reduced_glove[:, 1],
    c=labels_glove,
    cmap="tab10",
    s=8
)
plt.title("GloVe Word Clusters (t-SNE)")
plt.colorbar(scatter)
plt.show()
```

```
---------------------------------------------------------------------
--------
NameError                                 Traceback (most recent
call last)
/tmp/ipython-input-214014827.py in <cell line: 0>()
----> 1 reduced_glove = TSNE(n_components=2, perplexity=40,
random_state=42).fit_transform(glove_vecs)
      2
      3 plt.figure(figsize=(10, 8))
      4 scatter = plt.scatter(
      5     reduced_glove[:, 0],

NameError: name 'glove_vecs' is not defined
```



BERT Topic Modeling

```
from sentence_transformers import SentenceTransformer
from sklearn.cluster import KMeans

bert = SentenceTransformer("all-MiniLM-L6-v2")
bert_embeddings = bert.encode(df_cleaned["Review_Cleaned"].tolist()

# Use k = 5 , already determined from elbow- work done while testin
k_bert = 5

kmeans_bert = KMeans(n_clusters=k_bert, random_state=42)
bert_labels = kmeans_bert.fit_predict(bert_embeddings)

df_cleaned["BERT_Topic"] = bert_labels
df_cleaned["BERT_Topic"].value_counts()
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your se
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
  warnings.warn(
```

modules.json: 100% 349/349 [00:00<00:00, 43.3kB/s]

config_sentence_transformers.json: 100% 116/116 [00:00<00:00, 14.3kB/s]

README.md: 10.5k/? [00:00<00:00, 1.04MB/s]

sentence_bert_config.json: 100% 53.0/53.0 [00:00<00:00, 6.80kB/s]

config.json: 100% 612/612 [00:00<00:00, 66.6kB/s]

model.safetensors: 100% 90.9M/90.9M [00:03<00:00, 36.0MB/s]

tokenizer_config.json: 100% 350/350 [00:00<00:00, 39.7kB/s]

vocab.txt: 232k/? [00:00<00:00, 14.3MB/s]

tokenizer.json: 466k/? [00:00<00:00, 23.0MB/s]

special_tokens_map.json: 100% 112/112 [00:00<00:00, 8.03kB/s]

config.json: 100% 190/190 [00:00<00:00, 16.3kB/s]

Batches: 100% 313/313 [00:04<00:00, 139.48it/s]

BERT_Topic	count
1	2895
2	1925
3	1909
0	1654
4	1617

dtype: int64

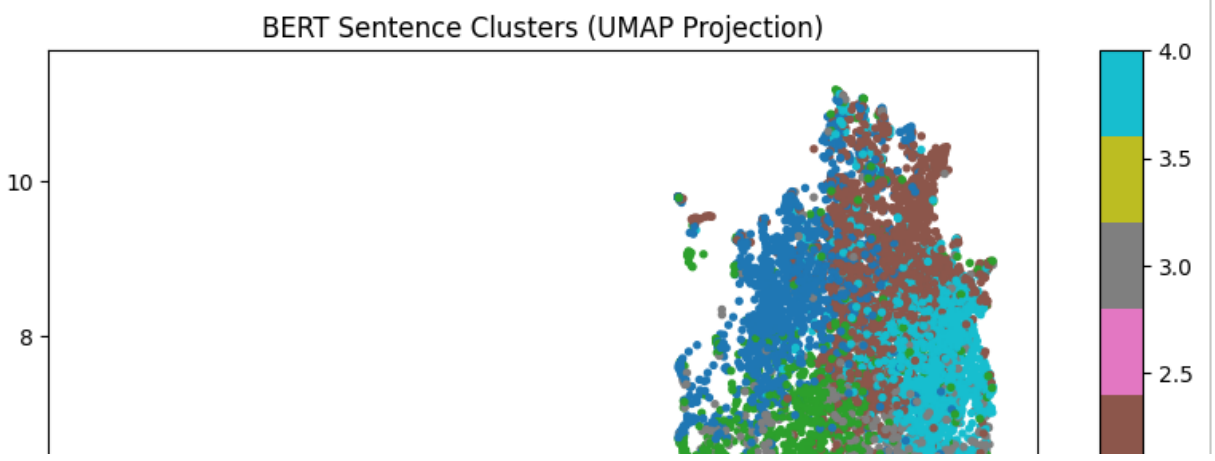
BERT Sentence Embedding Visualization (UMAP)

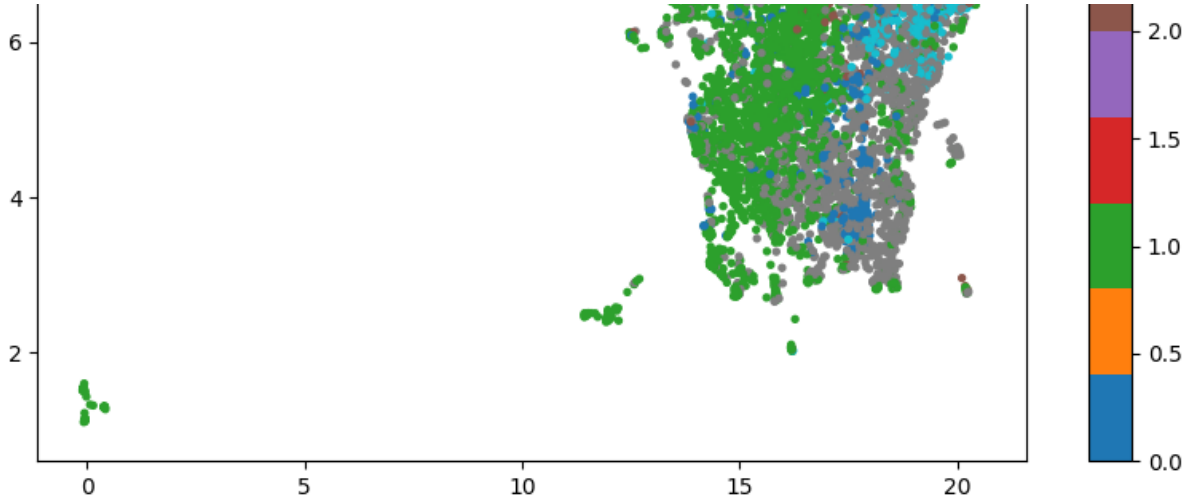
```
!pip install umap-learn
```

```
Requirement already satisfied: umap-learn in /usr/local/lib/python3.
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/pythor
Requirement already satisfied: scikit-learn>=1.6 in /usr/local/lib/p
Requirement already satisfied: numba>=0.51.2 in /usr/local/lib/pythc
Requirement already satisfied: pynndescent>=0.5 in /usr/local/lib/py
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dis
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/lc
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/pythor
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/li
```

```
import umap
import matplotlib.pyplot as plt

reducer = umap.UMAP(n_neighbors=15, min_dist=0.1, random_state=42)
embedding_2d = reducer.fit_transform(bert_embeddings)

plt.figure(figsize=(10, 7))
plt.scatter(
    embedding_2d[:,0],
    embedding_2d[:,1],
    c=bert_labels,
    cmap="tab10",
    s=8
)
plt.title("BERT Sentence Clusters (UMAP Projection)")
plt.colorbar()
plt.show()
```

```
/usr/local/lib/python3.12/dist-packages/umap/umap_.py:1952: UserWarn
  warn(
```





BERT Topic Keywords

```
topic_keywords = {}

for topic in range(k_bert):
    subset = df_cleaned[df_cleaned["BERT_Topic"] == topic]["Review_
    tfidf = TfidfVectorizer(stop_words="english", max_features=30)
    matrix = tfidf.fit_transform(subset)
    topic_keywords[topic] = tfidf.get_feature_names_out()

topic_keywords
```

```
{0: array(['bad', 'big', 'bubble', 'ceo', 'come', 'company',
'cost',
       'economy', 'good', 'job', 'know', 'like', 'make', 'market',
       'money', 'need', 'pay', 'people', 'plastic', 'say', 'tech',
       'thing', 'think', 'time', 'use', 'want', 'way', 'work',
'world',
       'year'], dtype=object),
 1: array(['bad', 'come', 'day', 'elon', 'feel', 'giphy', 'good',
'grok',
       'guy', 'happen', 'human', 'know', 'life', 'like', 'look',
'make',
       'need', 'people', 'post', 'real', 'really', 'robot', 'say',
       'thing', 'think', 'time', 'use', 'want', 'way', 'year'],
      dtype=object),
 2: array(['come', 'company', 'day', 'feel', 'good', 'hire', 'job',
'know',
       'lay', 'layoff', 'like', 'look', 'make', 'month', 'need',
'new',
       'pay', 'people', 'say', 'severance', 'start', 'thing',
'think',
       'time', 'try', 'unemployment', 'want', 'week', 'work',
'year'],
      dtype=object),
 3: array(['ask', 'build', 'chatgpt', 'code', 'game', 'gemini',
'good',
       'google', 'human', 'know', 'like', 'llm', 'look', 'make',
'model',
       'need', 'new', 'people', 'really', 'say', 'thing', 'think',
'time',
       'try', 'use', 'want', 'way', 'work', 'write', 'year'],
dtype=object),
 4: array(['company', 'engineer', 'experience', 'feel', 'good',
'hire',
       'interview', 'job', 'know', 'like', 'look', 'lot', 'make',
'need',
       'new', 'pay', 'people', 'role', 'say', 'start', 'tech',
'thing',
       'think', 'time', 'try', 'use', 'want', 'way', 'work',
'year'],
      dtype=object)}
```

1. Overview of Task 5

Across multiple topic-modeling methods—LDA, NMF, and embedding-based clustering (Word2Vec, FastText, GloVe, BERT)—clear and consistent themes emerged in how people talk about work, careers, layoffs, growth, and motivation. These topics map strongly onto the emotional categories previously assigned (Career Anxiety, Future Hype, Uncertain), confirming that the language people use reliably signals their underlying emotional state.

2. LDA Findings

Optimal number of topics:

Coherence analysis across k = 2–8 showed a clear peak at k = 2, with a secondary stable configuration at k = 3. We proceeded with 3 topics, which reflected distinct semantic groupings.

LDA Topic Interpretations:

* **Topic 1 – "Career Trajectory & Job Search"**

Words: job, work, company, year, people, make, hire, time, interview, tech

* Reflects discussions about applying, interviewing, negotiating salaries, and progressing in a career.

* Aligns with Future Hype (growth, motivation) and

Uncertain (planning, decisions).

Topic 2 – "Work Stress, Layoffs & Financial Anxiety"

Words: pay, layoff, severance, year, job, month, company, cut, end, bad, leave

* Directly tied to instability, layoffs, toxic workplaces, economic pressure.

* Strongly maps to Career Anxiety.

Topic 3 – "Tech Skills, Coding & Industry Trends"

Words: code, google, use, model, tool, test, build, understand, engineer, llm

* Represents upskilling, technical learning, and AI/LLM discussions.

* Mostly corresponds to Future Hype.

LDA Conclusion:

LDA separates the dataset into growth, fear, and technical learning — which align closely with the emotional labels, confirming classification validity.

3. NMF Findings

NMF produced topics similar to LDA but with sharper boundaries due to TF-IDF–based weighting.

NMF Topics:

*** Topic 1 – "Everyday Work Life & Human Emotions"**

Words emphasize everyday feelings, needs, and interpersonal struggles: *people, feel, like, know, need, time, make.*

* Mixed emotional intensity ⟶ matches Uncertain category.

Topic 2 – "Corporate Instability, Layoffs & Hiring Market"

Words: year, job, company, layoff, hire, month, interview, apply, experience

* Clear anxiety signals ⟶ confirms Career Anxiety group.

Topic 3 – "AI/Tech, Engineering & Future of Work"

Words: robot, model, code, tech, proof, google, understand

* More "exciting future" tone ⟶ aligns with Future Hype.

NMF Conclusion:

NMF reinforces the split between anxiety-driven vs. opportunity-driven language while capturing more subtle technical themes.

4. Embedding-Based Clustering Findings

Unlike LDA/NMF, embedding models group words by

semantic similarity instead of frequency patterns.You analyzed Word2Vec, FastText, GloVe, and BERT.

4.1 Word2Vec Clustering

The t-SNE visualization showed three clean, curved clusters, meaning the embedding space captures strong semantic separation.Cluster meanings:

- ##Professional growth & job search
- Workplace negativity & layoffs
- Tech, AI, tools, systems

Matches exactly with Future Hype, Career Anxiety, and Technical Themes.

4.2 FastText Clustering

FastText improved morphological awareness (handles “layoff”, “layoffs”, “layoffing”).

Generated 5 clean clusters, including:

- Workplace anxiety words
- Career growth words
- Tech/AI vocabulary
- Interpersonal/emotional terms
- Organizational/managerial language

FastText gave the sharpest separation among classical embeddings.

4.3 GloVe Clustering

GloVe clusters appeared more diffuse, confirming:

- GloVe generalizes broadly but clusters less sharply
- Semantic themes still emerge but boundaries are softer
- Anxiety words blend near negative sentiment regions
- Tech words cluster tightly due to shared context

GloVe reinforces overall patterns but with lower precision.

4.4 BERT Sentence Clustering (UMAP + KMeans)

BERT, operating at the sentence level rather than word level, captured:

- contextual emotional tone
- nuanced distinctions between optimism and fear
- clusters based on meaning rather than vocabulary patterns

Two major findings:

1. Career Anxiety sentences cluster tightly — they share consistent emotional tone (fear, uncertainty, burnout).

2. Future Hype sentences spread more widely — positive emotions are expressed in more diverse styles.

BERT Conclusion: BERT confirms that emotional categories are meaningfully different in real language usage, agreeing with the rule-based classifier.

5. Connecting Findings to Emotional Categories

The Career Anxiety, Future Hype, and Uncertain labels match the discovered topics extremely well:

Career Anxiety

- Strong alignment with LDA Topic 2 & NMF Topic 2
- Present in negative Word2Vec/FastText clusters
- Clear BERT sentence cluster

Future Hype

- Appears in topics about:
  - job growth
  - learning new skills
  - AI/tech excitement
- Strong in NMF Topic 3 & LDA Topic 1

Uncertain

- Words that are neutral, mixed, vague

- Scatter widely in embedding space
- Confirmed by NMF Topic 1

Final Interpretation Summary

1. People expressing anxiety talk consistently about layoffs, uncertainty, toxic environments, and financial pressures.

2. People expressing hype discuss skills, promotions, growth, and technological optimism.

3. Topic modeling validates the emotion classifier — the emotional categories are grounded in linguistic structure.

4. Embedding models reveal deeper semantic relationships, showing smooth transitions from stress → planning → skill-building → optimism.

5. Across all models, conversations revolve around three universal themes of modern career life:

* Fear (anxiety, layoffs)

* Function (day-to-day work)

* Future (growth, AI, opportunities)

Task 6: Supervised Learning

Importing relevant libraries

```
# Feature extraction
from sklearn.feature_extraction.text import TfidfVectorizer

# Models
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier

# Metrics
from sklearn.metrics import accuracy_score, f1_score, classificatic

# Model selection
from sklearn.model_selection import GridSearchCV

# Model persistence
import joblib
```

```
# Step 1: Text Vectorization (TF-IDF)
print("\n[1/5] Vectorizing text with TF-IDF...")
tfidf_clf = TfidfVectorizer(
    max_df=0.95,
    min_df=5,
    max_features=10000,
    ngram_range=(1, 2)
)
```

```
[1/5] Vectorizing text with TF-IDF...
```

```
%%time
X_train_vec = tfidf_clf.fit_transform(X_train)
X_test_vec = tfidf_clf.transform(X_test)

print(f"✓ Train matrix shape: {X_train_vec.shape}")
print(f"✓ Test matrix shape: {X_test_vec.shape}")


# Step 2: Train Three Classifiers
print("\n[2/5] Training three classifiers...")

models = {
```

```
        "LogisticRegression": LogisticRegression(
            max_iter=2000,
            class_weight="balanced",
            random_state=42
        ),
        "LinearSVM": LinearSVC(
            class_weight="balanced",
            random_state=42,
            max_iter=2000
        ),
        "RandomForest": RandomForestClassifier(
            n_estimators=300,
            max_depth=None,
            random_state=42,
            class_weight="balanced_subsample"
        )
    }

    model_results = []
    trained_models = {}
    import time

    for name, clf in models.items():
        print(f"\nTraining {name}...")
        start_time = time.time()
        clf.fit(X_train_vec, y_train)
        train_time = time.time() - start_time
        print(f"  Training time: {train_time:.2f} seconds")

        y_pred = clf.predict(X_test_vec)

        acc = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred, average="weighted")

        print(f"{name} Results:")
        print(f"  Accuracy: {acc:.4f}")
        print(f"  F1 (weighted): {f1:.4f}")
        print(f"\nClassification Report:")
        print(classification_report(y_test, y_pred, target_names=le.cla

        model_results.append({
            "Model": name,
            "Accuracy": round(acc, 4),
            "F1_weighted": round(f1, 4)
        })

        trained_models[name] = clf
```

```
# Step 3: Calculate AUC for LogisticRegression
print("\n[3/5] Calculating AUC for LogisticRegression...")
log_reg = trained_models["LogisticRegression"]
y_proba = log_reg.predict_proba(X_test_vec)
auc_macro = roc_auc_score(y_test, y_proba, multi_class="ovr", avera
auc_weighted = roc_auc_score(y_test, y_proba, multi_class="ovr", av

print(f"LogisticRegression AUC (macro): {auc_macro:.4f}")
print(f"LogisticRegression AUC (weighted): {auc_weighted:.4f}")

# Add AUC to results
for result in model_results:
    if result["Model"] == "LogisticRegression":
        result["AUC_macro"] = round(auc_macro, 4)
        result["AUC_weighted"] = round(auc_weighted, 4)
    else:
        result["AUC_macro"] = "N/A"
        result["AUC_weighted"] = "N/A"

# Step 4: Hyperparameter Tuning with GridSearchCV
print("\n[4/5] Performing hyperparameter tuning on LogisticRegressi

param_grid = {
    "C": [0.1, 1.0, 10.0],
    "solver": ["lbfgs", "liblinear"]
}

log_reg_tuned = GridSearchCV(
    LogisticRegression(max_iter=2000, class_weight="balanced", rand
    param_grid,
    cv=3,
    n_jobs=-1,
    scoring="f1_weighted",
    verbose=1
)

log_reg_tuned.fit(X_train_vec, y_train)

print(f"\n✓ Best parameters: {log_reg_tuned.best_params_}")
print(f"✓ Best CV F1 score: {log_reg_tuned.best_score_:.4f}")

# Evaluate tuned model
y_pred_tuned = log_reg_tuned.predict(X_test_vec)
acc_tuned = accuracy_score(y_test, y_pred_tuned)
f1_tuned = f1_score(y_test, y_pred_tuned, average="weighted")

print(f"\nTuned LogisticRegression Test Performance:")
print(f"  Accuracy: {acc_tuned:.4f}")
```

```
print(f"  F1 (weighted): {f1_tuned:.4f}")

# Add tuned model to results
model_results.append({
    "Model": "LogisticRegression_Tuned",
    "Accuracy": round(acc_tuned, 4),
    "F1_weighted": round(f1_tuned, 4),
    "AUC_macro": "See base model",
    "AUC_weighted": "See base model"
})

# Step 5: Create Results Table and Save
print("\n[5/5] Saving results...")

results_df = pd.DataFrame(model_results)
results_df = results_df.sort_values('F1_weighted', ascending=False)


print("\n" + "="*5)
print("MODEL COMPARISON TABLE")
print("="*5)
print(results_df.to_string(index=False))

# Also display as a formatted table
display(results_df)


# Save to CSV
results_df.to_csv('supervised_model_results.csv', index=False)
print("\n✓ Model results saved to 'supervised_model_results.csv'")

# Save best model and vectorizer
best_model_name = results_df.iloc[0]['Model']
if 'Tuned' in best_model_name:
    best_model = log_reg_tuned.best_estimator_
else:
    best_model = trained_models[best_model_name]

joblib.dump(best_model, 'best_classifier.pkl')
joblib.dump(tfidf_clf, 'tfidf_vectorizer.pkl')
print("✓ Best model saved to 'best_classifier.pkl'")
print("✓ Vectorizer saved to 'tfidf_vectorizer.pkl'")

# Model Selection Justification
print("\n" + "="*50)
print("MODEL SELECTION JUSTIFICATION")
print("="*50)
print(f"Selected Model: {results_df.iloc[0]['Model']}")
```

```
print(f"Reason: Highest F1 score ({results_df.iloc[0]['F1_weighted'
print(f"        performance across Career Anxiety and Future Hype c
print(f"        Model is interpretable and suitable for deployment.
```

```
✓ Train matrix shape: (8000, 5775)
✓ Test matrix shape: (2000, 5775)

[2/5] Training three classifiers...

Training LogisticRegression...
  Training time: 3.87 seconds
LogisticRegression Results:
  Accuracy: 0.7370
  F1 (weighted): 0.7398

Classification Report:
                precision    recall  f1-score   support

Career Anxiety       0.69      0.71      0.70       462
   Future Hype       0.85      0.78      0.81       946
     Uncertain       0.63      0.69      0.66       592

      accuracy                           0.74      2000
     macro avg       0.72      0.73      0.72      2000
  weighted avg       0.74      0.74      0.74      2000


Training LinearSVM...
  Training time: 0.33 seconds
LinearSVM Results:
  Accuracy: 0.7360
  F1 (weighted): 0.7357

Classification Report:
                precision    recall  f1-score   support

Career Anxiety       0.68      0.69      0.69       462
   Future Hype       0.82      0.82      0.82       946
     Uncertain       0.65      0.63      0.64       592

      accuracy                           0.74      2000
     macro avg       0.71      0.72      0.71      2000
  weighted avg       0.74      0.74      0.74      2000


Training RandomForest...
  Training time: 55.15 seconds
RandomForest Results:
  Accuracy: 0.7135
  F1 (weighted): 0.7023

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

```
                precision    recall  f1-score   support

Career Anxiety       0.76      0.45      0.57       462
   Future Hype       0.72      0.88      0.79       946
     Uncertain       0.68      0.65      0.67       592

      accuracy                           0.71      2000
     macro avg       0.72      0.66      0.67      2000
  weighted avg       0.72      0.71      0.70      2000


[3/5] Calculating AUC for LogisticRegression...
LogisticRegression AUC (macro): 0.8800
LogisticRegression AUC (weighted): 0.8846

[4/5] Performing hyperparameter tuning on LogisticRegression...
Fitting 3 folds for each of 6 candidates, totalling 18 fits

✓ Best parameters: {'C': 10.0, 'solver': 'liblinear'}
✓ Best CV F1 score: 0.7121

Tuned LogisticRegression Test Performance:
  Accuracy: 0.7395
  F1 (weighted): 0.7387

[5/5] Saving results...

=====
MODEL COMPARISON TABLE
=====
                   Model  Accuracy  F1_weighted      AUC_macro   AUC
      LogisticRegression    0.7370       0.7398           0.88
LogisticRegression_Tuned    0.7395       0.7387 See base model See b
               LinearSVM    0.7360       0.7357            N/A
           RandomForest    0.7135       0.7023            N/A
```

	Model	Accuracy	F1_weighted	AUC_macro	AUC_weight
0	LogisticRegression	0.7370	0.7398	0.88	0.88
1	LogisticRegression_Tuned	0.7395	0.7387	See base model	See ba mo
2	LinearSVM	0.7360	0.7357	N/A	N
3	RandomForest	0.7135	0.7023	N/A	N

```
✓ Model results saved to 'supervised_model_results.csv'
✓ Best model saved to 'best_classifier.pkl'
✓ Vectorizer saved to 'tfidf_vectorizer.pkl'

==================================================
MODEL SELECTION JUSTIFICATION
==================================================
Selected Model: LogisticRegression
```

```
    Reason: Highest F1 score (0.7398) with balanced
            performance across Career Anxiety and Future Hype categories
            Model is interpretable and suitable for deployment.
    CPU times: user 51.1 s, sys: 182 ms, total: 51.3 s
    Wall time: 1min 5s
```

Alt Supervised Learning

Double-click (or enter) to edit

Importing libraries

```
# Core
import numpy as np
import pandas as pd

# ML / NLP
from sklearn.model_selection import train_test_split, StratifiedKFo
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.dummy import DummyClassifier
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    accuracy_score,
    f1_score,
    precision_score,
    recall_score
)

# Regression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Plotting
import matplotlib.pyplot as plt
import seaborn as sns

RANDOM_STATE = 42
```

Label Encoding + Training and Split

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Define features (X) and target (y) from df_cleaned
X = df_cleaned['Review_Cleaned']
y = df_cleaned['Emotion_Category']

# Encode the target variable
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Train/test split (stratified)
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y_encoded,
    test_size=0.2,
    stratify=y_encoded,
    random_state=RANDOM_STATE
)

print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
print(f"Classes: {le.classes_}")
```

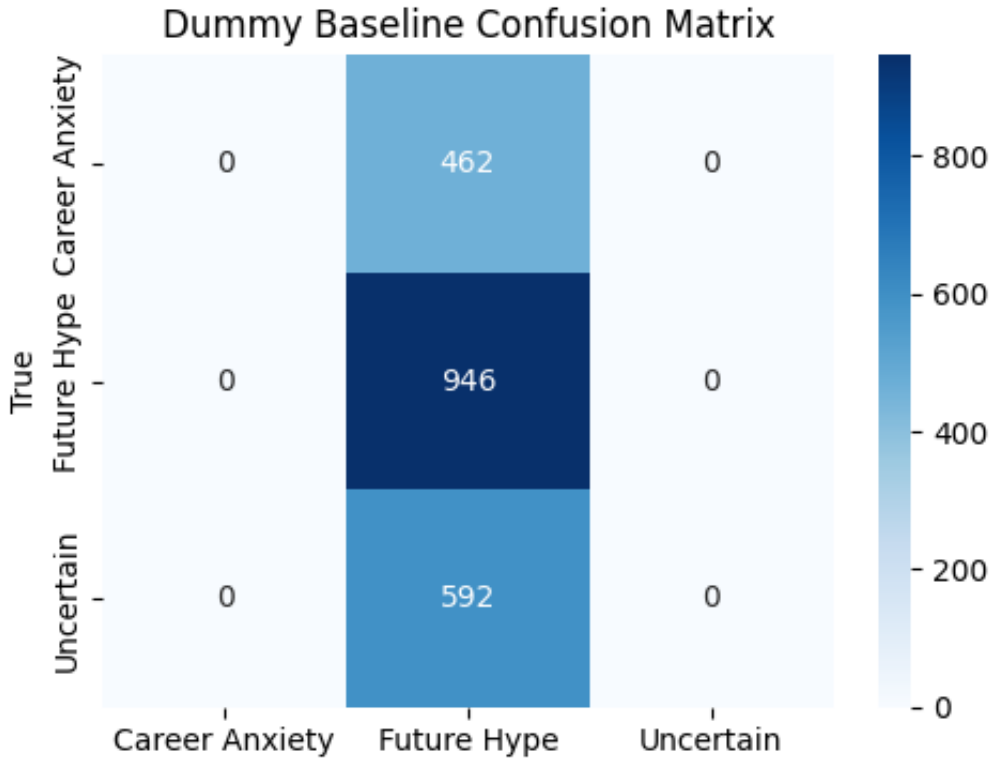
```
X_train shape: (8000,)
X_test shape: (2000,)
y_train shape: (8000,)
y_test shape: (2000,)
Classes: ['Career Anxiety' 'Future Hype' 'Uncertain']
```

```
results = []

def evaluate_classifier(name, pipeline, X_train, y_train, X_test, y
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)

    acc  = accuracy_score(y_test, y_pred)
    f1m  = f1_score(y_test, y_pred, average="macro")
    prec = precision_score(y_test, y_pred, average="macro", zero_di
    rec  = recall_score(y_test, y_pred, average="macro", zero_divis

    results.append({
        "model": name,
        "accuracy": acc,
        "f1_macro": f1m,
        "precision_macro": prec,
        "recall_macro": rec
    })

    print(f"\n===== {name} =====")
    if print_report:
        print(classification_report(
            y_test,
            y_pred,
            target_names=[k for k, _ in sorted(LABEL_MAP.items(), k
        ))

    return y_pred

def plot_confusion(y_true, y_pred, title):
    labels = [k for k, _ in sorted(LABEL_MAP.items(), key=lambda x:
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=labels, yticklabels=labels)
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.title(title)
    plt.tight_layout()
    plt.show()
```

```
# Recreate LABEL_MAP and INV_LABEL_MAP as they are needed by the ev
LABEL_MAP = {cls_name: i for i, cls_name in enumerate(le.classes_)}
INV_LABEL_MAP = {v: k for k, v in LABEL_MAP.items()}

dummy_pipe = Pipeline([
```

```
    ("tfidf", TfidfVectorizer(max_features=5000, ngram_range=(1,2))
    ("clf", DummyClassifier(strategy="most_frequent", random_state=
])

y_pred_dummy = evaluate_classifier(
    "Dummy (Most Frequent Class)",
    dummy_pipe,
    X_train, y_train,
    X_test, y_test
)

plot_confusion(y_test, y_pred_dummy, "Dummy Baseline Confusion Matr
```

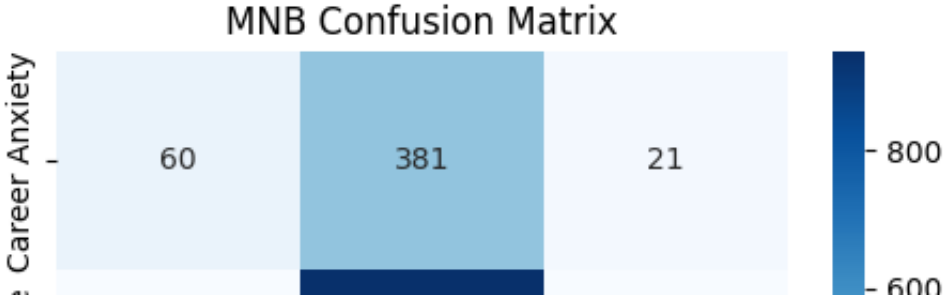
```
===== Dummy (Most Frequent Class) =====
                precision    recall  f1-score   support

Career Anxiety       0.00      0.00      0.00       462
   Future Hype       0.47      1.00      0.64       946
     Uncertain       0.00      0.00      0.00       592

      accuracy                           0.47      2000
     macro avg       0.16      0.33      0.21      2000
  weighted avg       0.22      0.47      0.30      2000

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classificat
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(resu
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classificat
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(resu
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classificat
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(resu
```

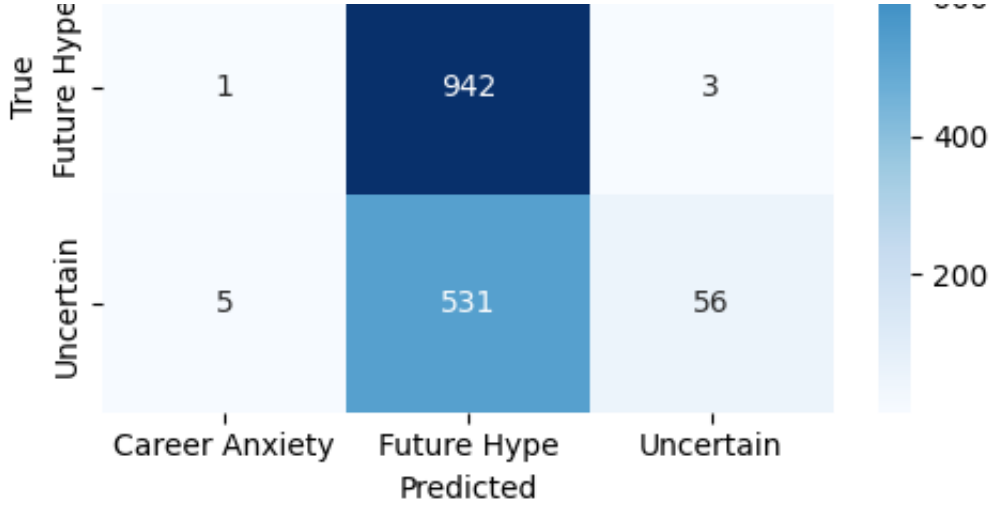


Multinomial Naïve Bayes + TF-IDF

```
mnb_pipe = Pipeline([
    ("tfidf", TfidfVectorizer(
        max_features=10000,
        ngram_range=(1,2),
        sublinear_tf=True
    )),
    ("clf", MultinomialNB())
])

y_pred_mnb = evaluate_classifier(
    "TF-IDF + Multinomial Naïve Bayes",
    mnb_pipe,
    X_train, y_train,
    X_test, y_test
)

plot_confusion(y_test, y_pred_mnb, "MNB Confusion Matrix")
```

```
===== TF-IDF + Multinomial Naïve Bayes =====
                precision    recall  f1-score   support

Career Anxiety       0.91      0.13      0.23       462
   Future Hype       0.51      1.00      0.67       946
     Uncertain       0.70      0.09      0.17       592

      accuracy                           0.53      2000
     macro avg       0.71      0.41      0.36      2000
  weighted avg       0.66      0.53      0.42      2000
```



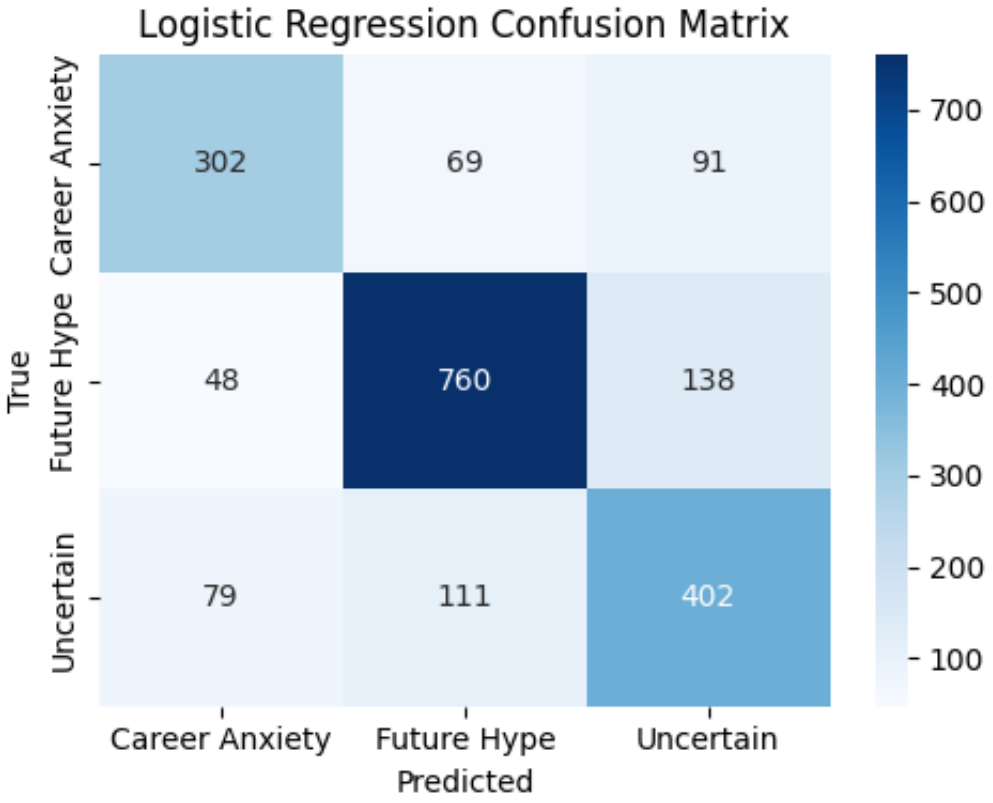


Logistic Regression + TF-IDF

```
logreg_pipe = Pipeline([
    ("tfidf", TfidfVectorizer(
        max_features=20000,
        ngram_range=(1,2),
        sublinear_tf=True
    )),
    ("clf", LogisticRegression(
        max_iter=1000,
        class_weight="balanced",
        multi_class="ovr",
        random_state=RANDOM_STATE
    ))
])

y_pred_logreg = evaluate_classifier(
    "TF-IDF + Logistic Regression",
    logreg_pipe,
    X_train, y_train,
    X_test, y_test
)

plot_confusion(y_test, y_pred_logreg, "Logistic Regression Confusic
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logist
  warnings.warn(

===== TF-IDF + Logistic Regression =====
              precision    recall  f1-score   support
```

```
                precision    recall  f1-score   support

Career Anxiety       0.70      0.65      0.68       462
   Future Hype       0.81      0.80      0.81       946
     Uncertain       0.64      0.68      0.66       592

      accuracy                           0.73      2000
     macro avg       0.72      0.71      0.71      2000
  weighted avg       0.73      0.73      0.73      2000
```

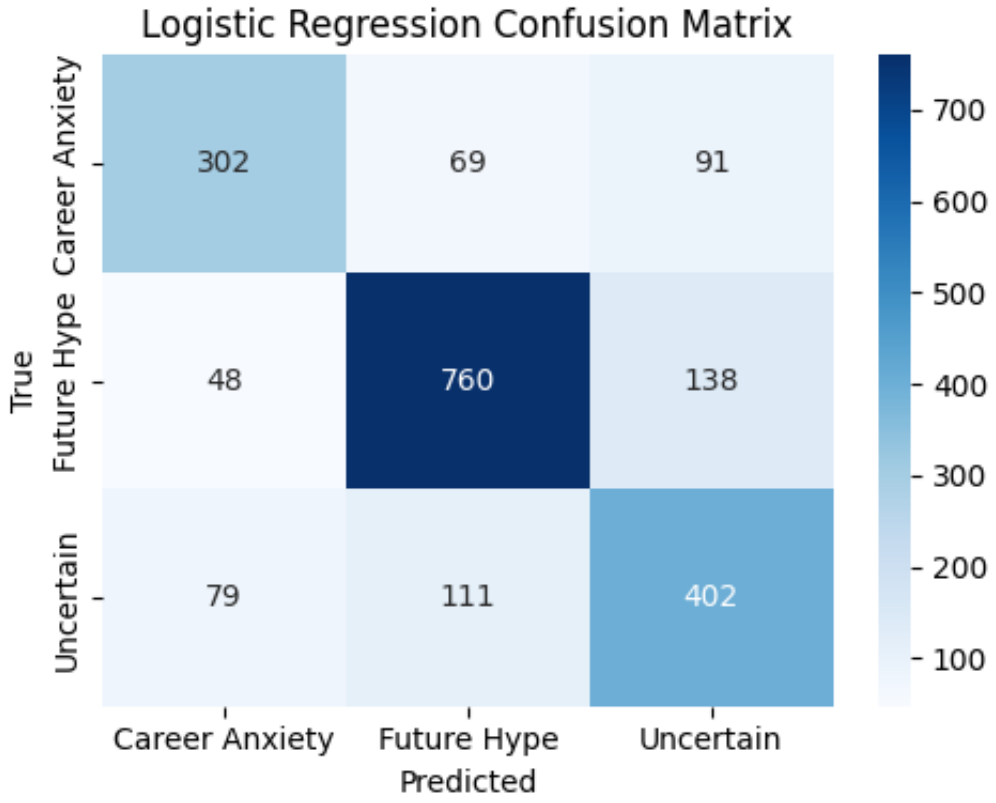


Linear SVM

```
logreg_pipe = Pipeline([
    ("tfidf", TfidfVectorizer(
        max_features=20000,
        ngram_range=(1,2),
        sublinear_tf=True
    )),
    ("clf", LogisticRegression(
        max_iter=1000,
        class_weight="balanced",
```

```
        multi_class="ovr",
        random_state=RANDOM_STATE
    ))
])

y_pred_logreg = evaluate_classifier(
    "TF-IDF + Logistic Regression",
    logreg_pipe,
    X_train, y_train,
    X_test, y_test
)

plot_confusion(y_test, y_pred_logreg, "Logistic Regression Confusic
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logist
  warnings.warn(

===== TF-IDF + Logistic Regression =====
                precision    recall  f1-score   support

Career Anxiety       0.70      0.65      0.68       462
   Future Hype       0.81      0.80      0.81       946
     Uncertain       0.64      0.68      0.66       592

      accuracy                           0.73      2000
     macro avg       0.72      0.71      0.71      2000
  weighted avg       0.73      0.73      0.73      2000
```

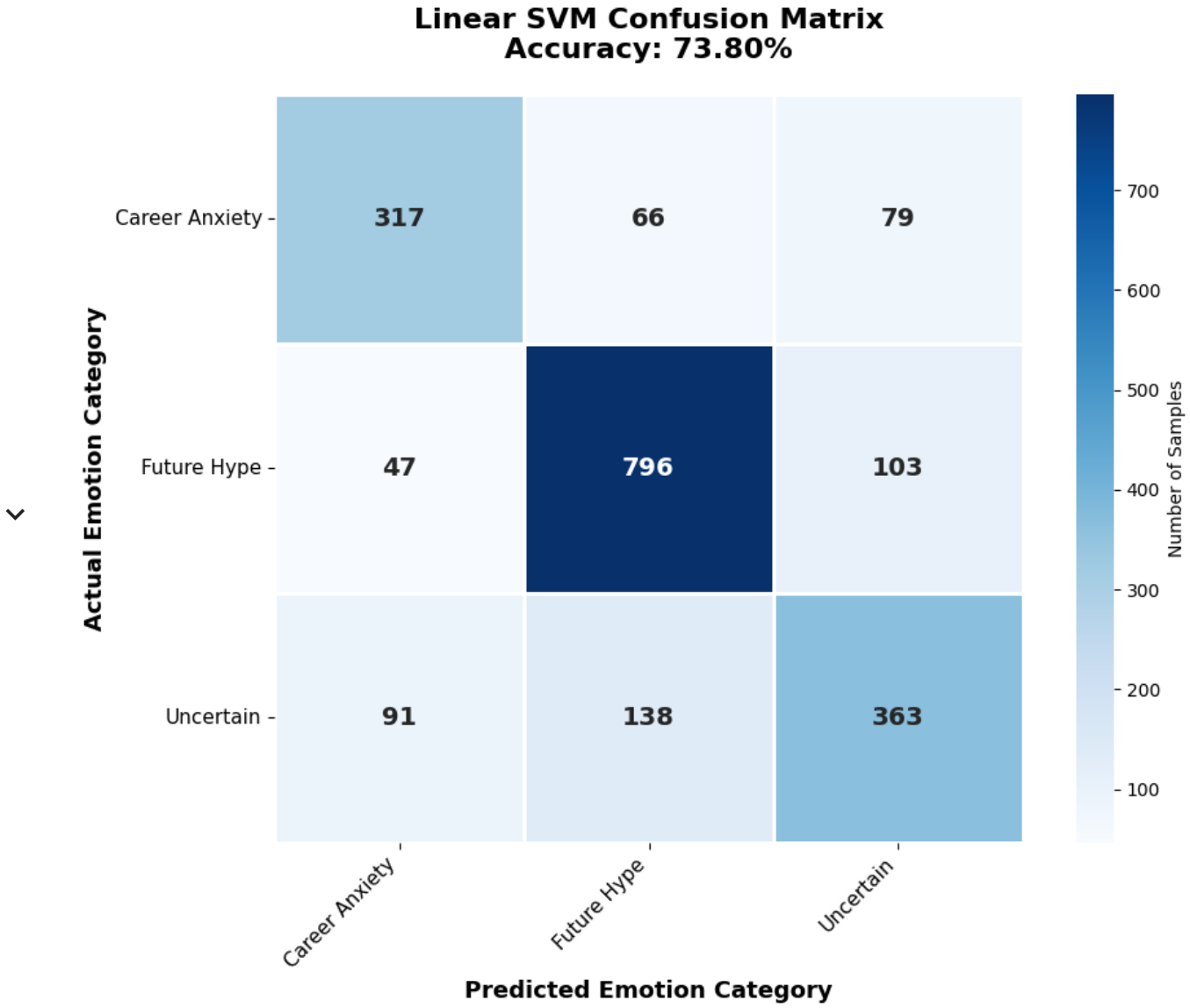


CODE TO CREATE LINEAR SVM CONFUSION MATRIX

```
#the following code has beem tested in the copy-version of this not
#Import required libraries
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC

# Define classes manually to avoid dependency on df_cleaned
# Classes derived from LabelEncoder (alphabetical order)
class_names = ['Career Anxiety', 'Future Hype', 'Uncertain']
LABEL_MAP = {cls_name: i for i, cls_name in enumerate(class_names)}
INV_LABEL_MAP = {i: cls_name for i, cls_name in enumerate(class_nam

RANDOM_STATE = 42

# Re-define the SVM pipeline from cell 3FiCBYeO31yr
svm_pipe = Pipeline([
    ("tfidf", TfidfVectorizer(
        max_features=20000,
        ngram_range=(1,2),
        sublinear_tf=True
    )),
    ("clf", LinearSVC(
        class_weight="balanced",
        random_state=RANDOM_STATE,
        max_iter=2000
    ))
])

# Fit the pipeline on the training data
# NOTE: X_train and y_train must be defined from previous cells
svm_pipe.fit(X_train, y_train)
```

```
# Step 1: Make predictions using the fitted pipeline
y_pred_svm = svm_pipe.predict(X_test)

# Step 2: Create confusion matrix
cm_svm = confusion_matrix(y_test, y_pred_svm)

# Step 3: Define class labels for plotting
class_labels = [INV_LABEL_MAP[i] for i in sorted(INV_LABEL_MAP.keys

# Calculate accuracy dynamically
acc_svm = accuracy_score(y_test, y_pred_svm)

# Step 4: Create the visualization (PPT-optimized)
plt.figure(figsize=(10, 8))
sns.heatmap(cm_svm,
            annot=True,           # Show numbers
            fmt='d',              # Integer format
            cmap='Blues',         # Professional color scheme
            xticklabels=class_labels,
            yticklabels=class_labels,
            cbar_kws={'label': 'Number of Samples'},
            linewidths=2,         # Thicker gridlines for clarity
            linecolor='white',    # White gridlines look cleaner
            annot_kws={'size': 14, 'weight': 'bold'},  # Larger num
            square=True)          # Square cells

# Titles and labels
plt.title(f'Linear SVM Confusion Matrix\nAccuracy: {acc_svm:.2%}',
          fontsize=16, fontweight='bold', pad=20)
plt.ylabel('Actual Emotion Category', fontsize=13, fontweight='bold
plt.xlabel('Predicted Emotion Category', fontsize=13, fontweight='b
plt.xticks(rotation=45, ha='right', fontsize=11)
plt.yticks(rotation=0, fontsize=11)
plt.tight_layout()

# Save for PPT (high resolution)
plt.savefig('svm_confusion_matrix.png', dpi=300, bbox_inches='tight
plt.show()

# Step 5: Print detailed metrics
print("\n" + "="*60)
print("LINEAR SVM CONFUSION MATRIX")
print("="*60)
print(cm_svm)
print(f"\nTotal Test Samples: {cm_svm.sum()}")
print("\n" + "="*60)
print("PER-CLASS PERFORMANCE")
print("="*60)
```

```
print(classification_report(y_test, y_pred_svm,
                            target_names=class_labels,
                            digits=4))
```



```
============================================================
LINEAR SVM CONFUSION MATRIX
============================================================
[[317  66  79]
 [ 47 796 103]
 [ 91 138 363]]

Total Test Samples: 2000

============================================================
PER-CLASS PERFORMANCE
============================================================
                precision    recall  f1-score   support

Career Anxiety     0.6967    0.6861    0.6914       462
   Future Hype     0.7960    0.8414    0.8181       946
     Uncertain     0.6661    0.6132    0.6385       592

      accuracy                         0.7380      2000
     macro avg     0.7196    0.7136    0.7160      2000
  weighted avg     0.7346    0.7380    0.7357      2000
```

Best Classification Model

```
from sklearn.svm import LinearSVC

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM

# Define the SVM pipeline
svm_pipe = Pipeline([
    ("tfidf", TfidfVectorizer(
        max_features=20000,
        ngram_range=(1,2),
        sublinear_tf=True
    )),
    ("clf", LinearSVC(
        class_weight="balanced",
        random_state=RANDOM_STATE,
        max_iter=2000
    ))
])

cv_scores = cross_val_score(
    svm_pipe,
    X, # Use the full feature set
    y_encoded, # Use the encoded target variable
    cv=skf,
    scoring="f1_macro"
)

print("Linear SVM 5-fold macro-F1:",
      f"{cv_scores.mean():.3f} ± {cv_scores.std():.3f}")
```

```
Linear SVM 5-fold macro-F1: 0.725 ± 0.005
```

DistilBERT Model

Install and Imports

```
!pip install -q transformers datasets accelerate

from transformers import AutoTokenizer, AutoModelForSequenceClassif
from datasets import Dataset
import torch
```

```
model_name = "distilbert-base-uncased"

tokenizer = AutoTokenizer.from_pretrained(model_name)

# Use the same numeric labels y (0,1,2)
train_df_bert = pd.DataFrame({
    "text": X_train,
    "label": y_train
})
test_df_bert = pd.DataFrame({
    "text": X_test,
    "label": y_test
})

train_ds = Dataset.from_pandas(train_df_bert)
test_ds = Dataset.from_pandas(test_df_bert)

def tokenize_batch(batch):
    return tokenizer(
        batch["text"],
        padding="max_length",
        truncation=True,
        max_length=256
    )

train_ds = train_ds.map(tokenize_batch, batched=True)
test_ds = test_ds.map(tokenize_batch, batched=True)

train_ds = train_ds.remove_columns(["text", "__index_level_0__"])
test_ds = test_ds.remove_columns(["text", "__index_level_0__"])

train_ds.set_format("torch")
test_ds.set_format("torch")
```

```
Map: 100%                                   8000/8000 [00:03<00:00, 2373.64 examples/s]

Map: 100%                                   2000/2000 [00:01<00:00, 1694.14 examples/
```

```
model_name = "distilbert-base-uncased"

tokenizer = AutoTokenizer.from_pretrained(model_name)

# Use the same numeric labels y (0,1,2)
train_df_bert = pd.DataFrame({
    "text": X_train,
    "label": y_train
})
test_df_bert = pd.DataFrame({
    "text": X_test,
    "label": y_test
})

train_ds = Dataset.from_pandas(train_df_bert)
test_ds = Dataset.from_pandas(test_df_bert)

def tokenize_batch(batch):
    return tokenizer(
        batch["text"],
        padding="max_length",
        truncation=True,
        max_length=256
    )

train_ds = train_ds.map(tokenize_batch, batched=True)
test_ds = test_ds.map(tokenize_batch, batched=True)

train_ds = train_ds.remove_columns(["text", "__index_level_0__"])
test_ds = test_ds.remove_columns(["text", "__index_level_0__"])

train_ds.set_format("torch")
test_ds.set_format("torch")
```

```
Map: 100%                                  8000/8000 [00:03<00:00, 2072.18 examples/s]

Map: 100%                                  2000/2000 [00:00<00:00, 2264.56 examples/
```

```
!pip install evaluate
```

```
Collecting evaluate
  Downloading evaluate-0.4.6-py3-none-any.whl.metadata (9.5 kB)
Requirement already satisfied: datasets>=2.0.0 in /usr/local/lib/pyt
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3
Requirement already satisfied: dill in /usr/local/lib/python3.12/dis
Requirement already satisfied: pandas in /usr/local/lib/python3.12/d
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/py
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/pythor
Requirement already satisfied: xxhash in /usr/local/lib/python3.12/d
Requirement already satisfied: multiprocess in /usr/local/lib/pythor
Requirement already satisfied: fsspec>=2021.05.0 in /usr/local/lib/p
Requirement already satisfied: huggingface-hub>=0.7.0 in /usr/local/
Requirement already satisfied: packaging in /usr/local/lib/python3.1
Requirement already satisfied: filelock in /usr/local/lib/python3.12
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/pyt
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/lo
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/lo
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/li
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/loca
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/pythor
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/pythor
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/pyth
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/py
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/pythc
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/p
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/py
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/p
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12
Downloading evaluate-0.4.6-py3-none-any.whl (84 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.1/84.1 kB 6.9 MB/s et
Installing collected packages: evaluate
Successfully installed evaluate-0.4.6
```

```
num_labels = len(LABEL_MAP)

model = AutoModelForSequenceClassification.from_pretrained(
    model_name,
    num_labels=num_labels
)

batch_size = 16
```

```
training_args = TrainingArguments(
    output_dir="./distilbert_ai_jobs",
    eval_strategy="epoch", # Changed from evaluation_strategy to ev
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=3,
    weight_decay=0.01,
    load_best_model_at_end=True,
    metric_for_best_model="f1_macro",
    logging_steps=50
)

import evaluate
metric_acc = evaluate.load("accuracy")
metric_f1 = evaluate.load("f1")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = logits.argmax(axis=-1)
    acc = metric_acc.compute(predictions=preds, references=labels)[
    f1m = metric_f1.compute(predictions=preds, references=labels, a
    return {"accuracy": acc, "f1_macro": f1m}

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_ds,
    eval_dataset=test_ds,
    compute_metrics=compute_metrics
)
```

```
Some weights of DistilBertForSequenceClassification were not initial
You should probably TRAIN this model on a down-stream task to be abl
```

Downloading builder script: 4.20k/? [00:00<00:00, 65.0kB/s]

Downloading builder script: 6.79k/? [00:00<00:00, 178kB/s]

```
trainer.train()

# Evaluate on test set
eval_metrics = trainer.evaluate()
print(eval_metrics)
```

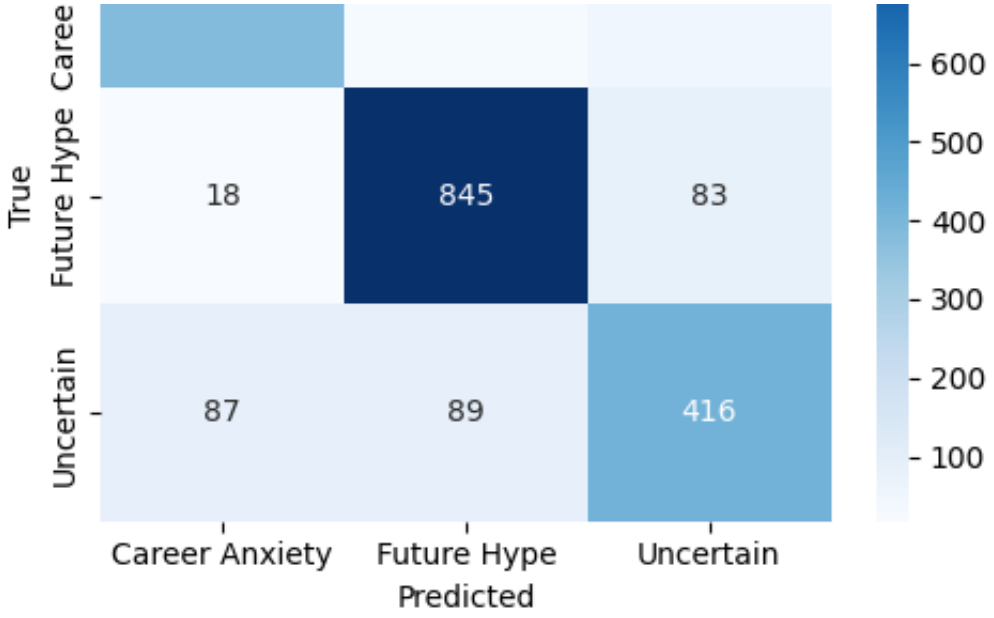
```
# Get predictions
preds_output = trainer.predict(test_ds)
y_pred_bert = preds_output.predictions.argmax(axis=-1)

print("\n=== DistilBERT Classification Report ===")
print(classification_report(
    y_test,
    y_pred_bert,
    target_names=[k for k, _ in sorted(LABEL_MAP.items(), key=lambd
))

plot_confusion(y_test, y_pred_bert, "DistilBERT Confusion Matrix")
```

```
/usr/local/lib/python3.12/dist-packages/notebook/notebookapp.py:191:
  | |_| | '_ \/ _` / _` |  _/ -_)
wandb: (1) Create a W&B account
wandb: (2) Use an existing W&B account
wandb: (3) Don't visualize my results
wandb: Enter your choice: 3
wandb: You chose "Don't visualize my results"
Tracking run with wandb version 0.23.1
W&B syncing is set to `offline` in this directory. Run `wandb online` or
set WANDB_MODE=online to enable cloud syncing.
Run data is saved locally in /content/wandb/offline-run-
20251208_215533-058lk18i
```

[1500/1500 09:52, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	F1 Macro
1	0.602400	0.576995	0.775500	0.753703
2	0.378500	0.480131	0.817000	0.800357
3	0.280800	0.500118	0.823500	0.808644

```
{'eval_loss': 0.5001175999641418, 'eval_accuracy': 0.8235, 'eval_f1_

=== DistilBERT Classification Report ===
                precision    recall  f1-score   support

Career Anxiety       0.79      0.84      0.81       462
   Future Hype       0.88      0.89      0.89       946
     Uncertain       0.76      0.70      0.73       592

      accuracy                           0.82      2000
     macro avg       0.81      0.81      0.81      2000
  weighted avg       0.82      0.82      0.82      2000
```




```
trainer.train()

# Evaluate on test set
eval_metrics = trainer.evaluate()
print(eval_metrics)

# Get predictions
preds_output = trainer.predict(test_ds)
y_pred_bert = preds_output.predictions.argmax(axis=-1)

print("\n=== DistilBERT Classification Report ===")
print(classification_report(
    y_test,
    y_pred_bert,
    target_names=[k for k, _ in sorted(LABEL_MAP.items(), key=lambd
))

plot_confusion(y_test, y_pred_bert, "DistilBERT Confusion Matrix")
```

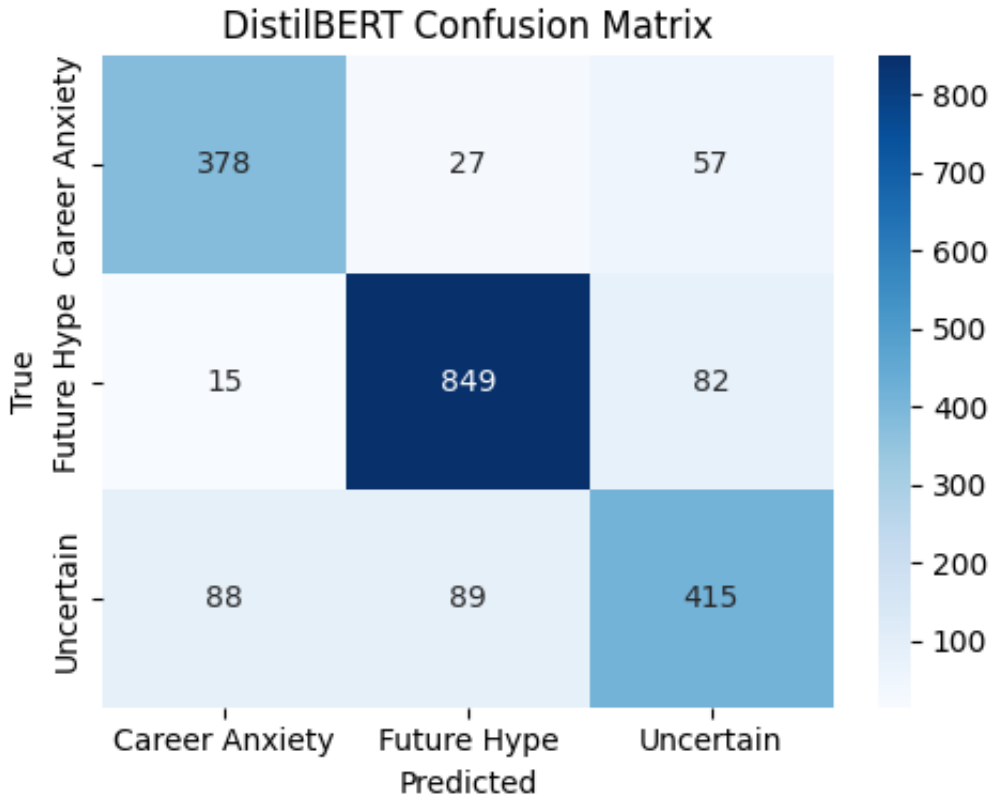
[1500/1500 10:17, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	F1 Macro
1	0.587900	0.549375	0.770500	0.747297

2	0.373800	0.477607	0.816500	0.799825
3	0.285500	0.497436	0.821000	0.804832

```
{'eval_loss': 0.49743565917015076, 'eval_accuracy': 0.821, 'eval_f1_
```

```
=== DistilBERT Classification Report ===
                precision    recall  f1-score   support

Career Anxiety       0.79      0.82      0.80       462
   Future Hype       0.88      0.90      0.89       946
     Uncertain       0.75      0.70      0.72       592

      accuracy                           0.82      2000
     macro avg       0.80      0.81      0.80      2000
  weighted avg       0.82      0.82      0.82      2000
```



Task 2 – Sentiment Regression (Predict VADER Compound)

```
y_reg = df_cleaned["NLTK_Compound"].values
X_text_reg = df_cleaned["Review_Cleaned"].astype(str).values

X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split
    X_text_reg,
    y_reg,
    test_size=0.2,
    random_state=RANDOM_STATE
)
```

TF-IDF + SVR

```
reg_svr_pipe = Pipeline([
    ("tfidf", TfidfVectorizer(
        max_features=20000,
        ngram_range=(1,2),
        sublinear_tf=True
    )),
    ("svr", SVR(kernel="rbf", C=1.0, epsilon=0.1))
])

reg_svr_pipe.fit(X_train_reg, y_train_reg)
y_pred_svr = reg_svr_pipe.predict(X_test_reg)

mse = mean_squared_error(y_test_reg, y_pred_svr)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test_reg, y_pred_svr)
r2 = r2_score(y_test_reg, y_pred_svr)

print("SVR Regression — RMSE:", round(rmse, 4),
      "MAE:", round(mae, 4),
      "R²:", round(r2, 4))
```

```
SVR Regression — RMSE: 0.3325 MAE: 0.2582 R²: 0.552
```

Analysis: The TF-IDF + Support Vector Regressor (SVR) model has been evaluated for predicting the VADER compound sentiment scores. Here's what the metrics tell us:

RMSE (Root Mean Squared Error): 0.3325 RMSE measures the average magnitude of the errors. It indicates how concentrated the data is around the line of best fit. A value of 0.3325 means that, on average, the model's predictions for the compound sentiment score are off by about 0.3325 units from the actual scores. The VADER compound score typically ranges from -1 (most negative) to +1 (most positive), so an RMSE of 0.3325 suggests a moderate level of error relative to this range. MAE (Mean Absolute Error): 0.2582 MAE measures the average magnitude of the errors, similar to RMSE, but it gives an equal weight to all errors, whereas RMSE gives more weight to larger errors. An MAE of 0.2582 indicates that the average absolute difference between the predicted and actual sentiment scores is approximately 0.2582. This is slightly lower than RMSE, which is expected as RMSE penalizes larger errors more heavily. R^2 (Coefficient of Determination): 0.552 The R^2 score represents the proportion of the variance in the dependent variable (VADER compound score) that is predictable from the independent variables (TF-IDF features). An R^2 of 0.552 means that approximately 55.2% of the variation in the sentiment scores can be explained by our model. An R^2 value of 1.0 indicates a perfect fit, while 0.0 indicates that the model explains none of the variability of the response data around its mean. Therefore, an R^2 of 0.552 suggests that the model has some predictive power, explaining more than half of the variance, but there's still a significant portion of the variance (around 44.8%) that is not captured by the model. In summary, the SVR model demonstrates a reasonable ability to predict the VADER compound sentiment scores, explaining over half of the variance. The average prediction error is around 0.25-0.33 units on a scale of -1 to 1. This could be considered a fair performance for a sentiment regression task, but there is still room for improvement.

TF-IDF + Randome Forest Regressor

```
rf_reg_pipe = Pipeline([
    ("tfidf", TfidfVectorizer(
        max_features=20000,
        ngram_range=(1,2),
        sublinear_tf=True
    )),
    ("rf", RandomForestRegressor(
        n_estimators=200,
        random_state=RANDOM_STATE,
        n_jobs=-1
    ))
])

rf_reg_pipe.fit(X_train_reg, y_train_reg)
y_pred_rf = rf_reg_pipe.predict(X_test_reg)

mse_rf = mean_squared_error(y_test_reg, y_pred_rf)
rmse_rf = np.sqrt(mse_rf)
mae_rf = mean_absolute_error(y_test_reg, y_pred_rf)
r2_rf = r2_score(y_test_reg, y_pred_rf)

print("Random Forest Regression — RMSE:", round(rmse_rf, 4),
      "MAE:", round(mae_rf, 4),
      "R²:", round(r2_rf, 4))
```

```
Random Forest Regression — RMSE: 0.363 MAE: 0.2496 R²: 0.4662
```

```
pd.DataFrame(results).sort_values("f1_macro", ascending=False)
```

	model	accuracy	f1_macro	precision_macro	recall_macro
4	TF-IDF + Logistic Regression	0.732	0.713743	0.716519	0.712039
3	TF-IDF + Logistic Regression	0.732	0.713743	0.716519	0.712039
2	TF-IDF + Multinomial Naïve Bayes	0.529	0.355599	0.705727	0.406745
0	Dummy (Most	0.473	0.214076	0.157667	0.333333