```
!pip -q install --upgrade transformers accelerate sentencepiece
wordcloud langdetect tqdm scikit-learn pyarrow
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/981.5 kB ? eta -:--:--
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 972.8/981.5 kB 60.5 MB/s eta
0:00:01 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 981.5/981.5 kB 18.9
MB/s eta 0:00:00
etadata (setup.py) ... ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
9.5/9.5 MB 63.9 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.8/42.8 MB 14.1 MB/s eta
0:00:00
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
pylibcudf-cu12 25.6.0 requires pyarrow<20.0.0a0,>=14.0.0;
platform_machine == "x86_64", but you have pyarrow 21.0.0 which is
incompatible.
cudf-cu12 25.6.0 requires pyarrow<20.0.0a0,>=14.0.0; platform_machine
== "x86_64", but you have pyarrow 21.0.0 which is incompatible.
```

```python
import os, math, gc, re, json, random
from datetime import datetime

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from tqdm import tqdm
import torch

from transformers import pipeline
from wordcloud import WordCloud
from langdetect import detect as lang_detect
from sklearn.feature_extraction.text import CountVectorizer

# Matplotlib defaults
plt.rcParams["figure.dpi"] = 120
plt.rcParams["axes.grid"] = False
```

Environment, Drive, Paths, Reproducibility

```python
from google.colab import drive
drive.mount('/content/drive')

# Persist HF cache so models don't re-download each session
os.environ["HF_HOME"] = "/content/drive/MyDrive/hf_cache"

# Project directories
BASE_DIR = "/content/drive/MyDrive/Final Project"
DATA_PATH = os.path.join(BASE_DIR, "news.tsv")          # MIND-small
```

```
RUN_DIR  = os.path.join(BASE_DIR, "run_outputs")         # chunk
outputs + full results
EXPORTS  = os.path.join(BASE_DIR, "exports")             # CSV/PNG
exports for sharing
os.makedirs(RUN_DIR, exist_ok=True)
os.makedirs(EXPORTS, exist_ok=True)

# Device
DEVICE = 0 if torch.cuda.is_available() else -1
print("□ Using GPU" if DEVICE == 0 else "⚠ Using CPU")

# Reproducibility
SEED = 42
random.seed(SEED); np.random.seed(SEED); torch.manual_seed(SEED)

# (Optional) Set this True only if you want to wipe previous run
outputs.
CLEAN_START = False
if CLEAN_START:
    for p in os.listdir(RUN_DIR):
        try:
            os.remove(os.path.join(RUN_DIR, p))
        except:
            pass
    print("Cleaned previous RUN_DIR files.")
```

```
Mounted at /content/drive
⚠ Using CPU
```

Config (batch sizes, truncation, chunking, weights)

```
CONFIG = {
    "max_length": 384,  # 256–384 is enough for abstracts
    "batch_sizes": {
        "sentiment": 32 if DEVICE == 0 else 8,
        "emotion":   24 if DEVICE == 0 else 8,
        "cred":      24 if DEVICE == 0 else 8,
        "bias":       6 if DEVICE == 0 else 2,  # zero-shot (heavy)
    },
    "chunk_size": 5000,          # process in 5k-row chunks (resume-
safe)
    "use_processed_text": True,  # True = use processed_text for
inference
    "weights": {"sent": 0.20, "emot": 0.20, "bias": 0.25, "cred":
0.35}
}
CONFIG
```

```
{'max_length': 384,
 'batch_sizes': {'sentiment': 32, 'emotion': 24, 'cred': 24, 'bias':
```

```
6},
 'chunk_size': 5000,
 'use_processed_text': True,
 'weights': {'sent': 0.2, 'emot': 0.2, 'bias': 0.25, 'cred': 0.35}}
```

Load MIND-small

```
cols =
["news_id","category","subcategory","title","abstract","url","title_en
tities","abstract_entities"]
data = pd.read_csv(DATA_PATH, sep="\t", header=None, names=cols)
print(f"Loaded: {data.shape[0]:,} rows")

data["title"] = data["title"].fillna("").astype(str)
data["abstract"] = data["abstract"].fillna("").astype(str)
data["combined_text"] = (data["title"] + " " +
data["abstract"]).str.strip()

display(data.head(3))
```

```
Loaded: 51,282 rows
```

{"summary":"{\n \"name\": \"display(data\",\n \"rows\": 3,\n \"fields\": [\n   {\n   \"column\": \"news_id\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"N55528\",\n   \"N19639\",\n   \"N61837\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"category\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"lifestyle\",\n   \"health\",\n   \"news\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"subcategory\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"lifestyleroyals\",\n   \"weightloss\",\n   \"newsworld\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"title\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"The Brands Queen Elizabeth, Prince Charles, and Prince Philip Swear By\",\n   \"50 Worst Habits For Belly Fat\",\n   \"The Cost of Trump's Aid Freeze in the Trenches of Ukraine's War\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"abstract\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"Shop the notebooks, jackets, and more that the royals can't live without.\",\n   \"These seemingly harmless habits are holding you back and keeping you from shedding that unwanted belly fat for good.\",\n   \"Lt. Ivan Molchanets

```
peeked over a parapet of sand bags at the front line of the war in
Ukraine. Next to him was an empty helmet propped up to trick snipers,
already perforated with multiple holes.\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"url\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n
\"https://assets.msn.com/labs/mind/AAGH0ET.html\",\n
\"https://assets.msn.com/labs/mind/AAB19MK.html\",\n
\"https://assets.msn.com/labs/mind/AAJgNsz.html\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"title_entities\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"[{\\\"Label\\\": \\\"Prince Philip, Duke of Edinburgh\\\",
\\\"Type\\\": \\\"P\\\", \\\"WikidataId\\\": \\\"Q80976\\\", \\\"Confi
dence\\\": 1.0, \\\"OccurrenceOffsets\\\": [48], \\\"SurfaceForms\\\":
[\\\"Prince Philip\\\"]}, {\\\"Label\\\": \\\"Charles, Prince of
Wales\\\", \\\"Type\\\": \\\"P\\\", \\\"WikidataId\\\": \\\"Q43274\\\"
, \\\"Confidence\\\": 1.0, \\\"OccurrenceOffsets\\\": [28],
\\\"SurfaceForms\\\": [\\\"Prince Charles\\\"]}, {\\\"Label\\\":
\\\"Elizabeth II\\\", \\\"Type\\\": \\\"P\\\", \\\"WikidataId\\\":
\\\"Q9682\\\", \\\"Confidence\\\": 0.97, \\\"OccurrenceOffsets\\\":
[11], \\\"SurfaceForms\\\": [\\\"Queen Elizabeth\\\"]}]\",\n
\"[{\\\"Label\\\": \\\"Adipose tissue\\\", \\\"Type\\\": \\\"C\\\",
\\\"WikidataId\\\": \\\"Q193583\\\", \\\"Confidence\\\": 1.0,
\\\"OccurrenceOffsets\\\": [20], \\\"SurfaceForms\\\": [\\\"Belly
Fat\\\"]}]\",\n          \"[]\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"abstract_entities\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"[]\",\n
\"[{\\\"Label\\\": \\\"Adipose tissue\\\", \\\"Type\\\": \\\"C\\\",
\\\"WikidataId\\\": \\\"Q193583\\\", \\\"Confidence\\\": 1.0,
\\\"OccurrenceOffsets\\\": [97], \\\"SurfaceForms\\\": [\\\"belly
fat\\\"]}]\",\n          \"[{\\\"Label\\\": \\\"Ukraine\\\",
\\\"Type\\\": \\\"G\\\", \\\"WikidataId\\\": \\\"Q212\\\", \\\"Confide
nce\\\": 0.946, \\\"OccurrenceOffsets\\\": [87], \\\"SurfaceForms\\\":
[\\\"Ukraine\\\"]}]\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"combined_text\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"The Brands Queen Elizabeth, Prince Charles, and Prince
Philip Swear By Shop the notebooks, jackets, and more that the royals
can't live without.\",\n          \"50 Worst Habits For Belly Fat
These seemingly harmless habits are holding you back and keeping you
from shedding that unwanted belly fat for good.\",\n          \"The
Cost of Trump's Aid Freeze in the Trenches of Ukraine's War Lt. Ivan
Molchanets peeked over a parapet of sand bags at the front line of the
```

```
war in Ukraine. Next to him was an empty helmet propped up to trick
snipers, already perforated with multiple holes.\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


Basic EDA: nulls, lengths, language, categories

```
# Nulls
print("Null counts:\n",
data.isna().sum().sort_values(ascending=False))

# Lengths
data["len_chars"] = data["combined_text"].str.len()
data["len_words"] = data["combined_text"].str.split().apply(len)
print("\nLength stats (chars):\n", data["len_chars"].describe())
print("\nLength stats (words):\n", data["len_words"].describe())

# Category overview
print("\nTop categories:\n", data["category"].value_counts().head(10))
print("\nTop subcategories:\n",
data["subcategory"].value_counts().head(10))

# Language spot check (100 items)
def safe_lang(s):
    s = (s or "").strip()
    if len(s) < 5: return "unk"
    try: return lang_detect(s[:400])
    except: return "unk"

lang_counts = data["combined_text"].sample(100,
random_state=SEED).apply(safe_lang).value_counts()
print("\nLanguage detection on sample(100):\n", lang_counts)
```

```
Null counts:
 abstract_entities    4
title_entities       3
news_id              0
category             0
subcategory          0
abstract             0
title                0
url                  0
combined_text        0
dtype: int64

Length stats (chars):
 count    51282.000000
mean       272.051071
std        160.229185
min         18.000000
```

```
25%        155.000000
50%        217.000000
75%        452.000000
max       2672.000000
Name: len_chars, dtype: float64

Length stats (words):
 count    51282.000000
mean        45.047736
std         26.843489
min          2.000000
25%         25.000000
50%         36.000000
75%         73.000000
max        485.000000
Name: len_words, dtype: float64

Top categories:
 category
news            15774
sports          14510
finance          3107
foodanddrink     2551
lifestyle        2479
travel           2350
video            2068
weather          2048
health           1885
autos            1639
Name: count, dtype: int64

Top subcategories:
 subcategory
newsus                       6564
football_nfl                 5420
newspolitics                 2826
newscrime                    2254
weathertopstories            2047
newsworld                    1720
football_ncaa                1665
baseball_mlb                 1661
basketball_nba               1555
newsscienceandtechnology     1210
Name: count, dtype: int64

Language detection on sample(100):
 combined_text
en    100
Name: count, dtype: int64
```

##Preprocessing

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize,
PunktSentenceTokenizer
from nltk.stem import WordNetLemmatizer

# Downloads (idempotent)
nltk.download('punkt'); nltk.download('stopwords');
nltk.download('wordnet')
try:
    nltk.download('punkt_tab')  # present in some Colab builds
except:
    pass

stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
_ = PunktSentenceTokenizer()

def preprocess(text: str) -> str:
    text = str(text)
    text = re.sub(r'\s+', ' ', text)
    text = re.sub(r'[^\w\s]', '', text)
    text = text.lower()
    sentences = sent_tokenize(text)
    tokens = []
    for sent in sentences:
        tokens.extend(word_tokenize(sent))
    tokens = [lemmatizer.lemmatize(w) for w in tokens if w not in
stop_words]
    return ' '.join(tokens)

PROC_PATH = os.path.join(RUN_DIR, "processed_text.parquet")
FORCE_REPROCESS = False   # set True if you want to recompute

if os.path.exists(PROC_PATH) and not FORCE_REPROCESS:
    print("Loading cached processed_text ...")
    proc_df = pd.read_parquet(PROC_PATH)
    data["processed_text"] = proc_df["processed_text"]
else:
    print("Preprocessing text ...")
    data["processed_text"] =
data["combined_text"].fillna("").astype(str).apply(preprocess)
    data[["processed_text"]].to_parquet(PROC_PATH, index=False)
    print("Saved:", PROC_PATH)

display(data[["combined_text","processed_text"]].head(3))
```

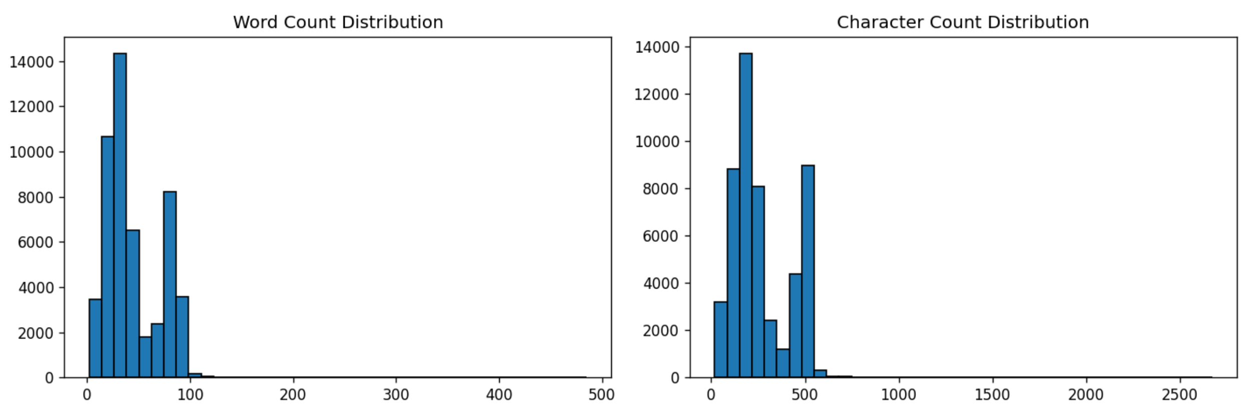
```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading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

Preprocessing text ...
Saved: /content/drive/MyDrive/Final
Project/run_outputs/processed_text.parquet
```


```
{"summary":"{\n  \"name\":
\"display(data[[\\\"combined_text\\\",\\\"processed_text\\\"]])\",\n
\"rows\": 3,\n  \"fields\": [\n    {\n      \"column\":
\"combined_text\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"The Brands Queen Elizabeth, Prince Charles, and Prince
Philip Swear By Shop the notebooks, jackets, and more that the royals
can't live without.\",\n          \"50 Worst Habits For Belly Fat
These seemingly harmless habits are holding you back and keeping you
from shedding that unwanted belly fat for good.\",\n          \"The
Cost of Trump's Aid Freeze in the Trenches of Ukraine's War Lt. Ivan
Molchanets peeked over a parapet of sand bags at the front line of the
war in Ukraine. Next to him was an empty helmet propped up to trick
snipers, already perforated with multiple holes.\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"processed_text\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"brand
queen elizabeth prince charles prince philip swear shop notebook
jacket royal cant live without\",\n          \"50 worst habit belly
fat seemingly harmless habit holding back keeping shedding unwanted
belly fat good\",\n          \"cost trump aid freeze trench ukraine
war lt ivan molchanets peeked parapet sand bag front line war ukraine
next empty helmet propped trick sniper already perforated multiple
hole\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

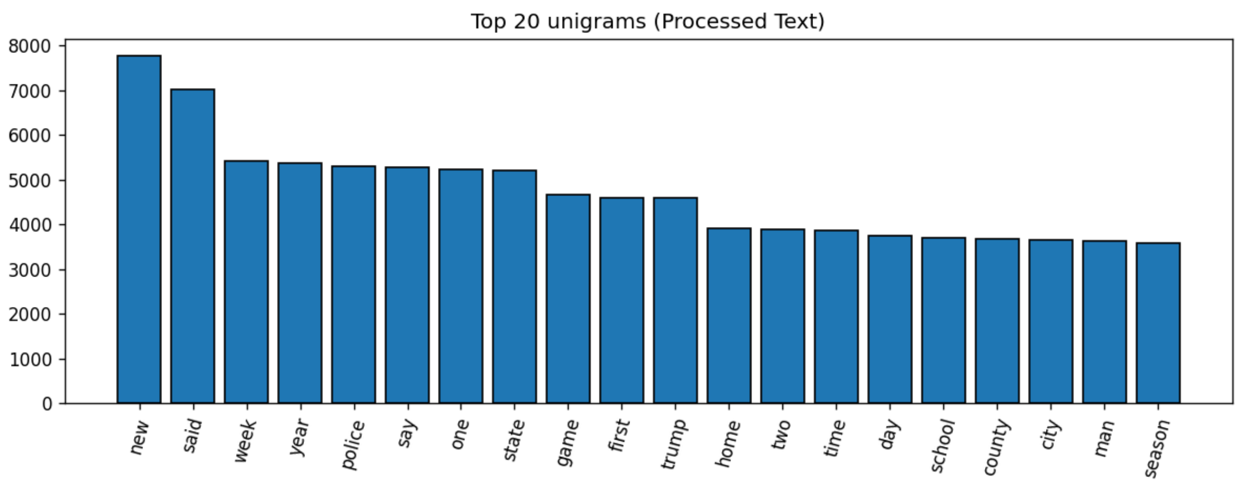

Quick EDA Visuals (lengths, word cloud, top n-grams)

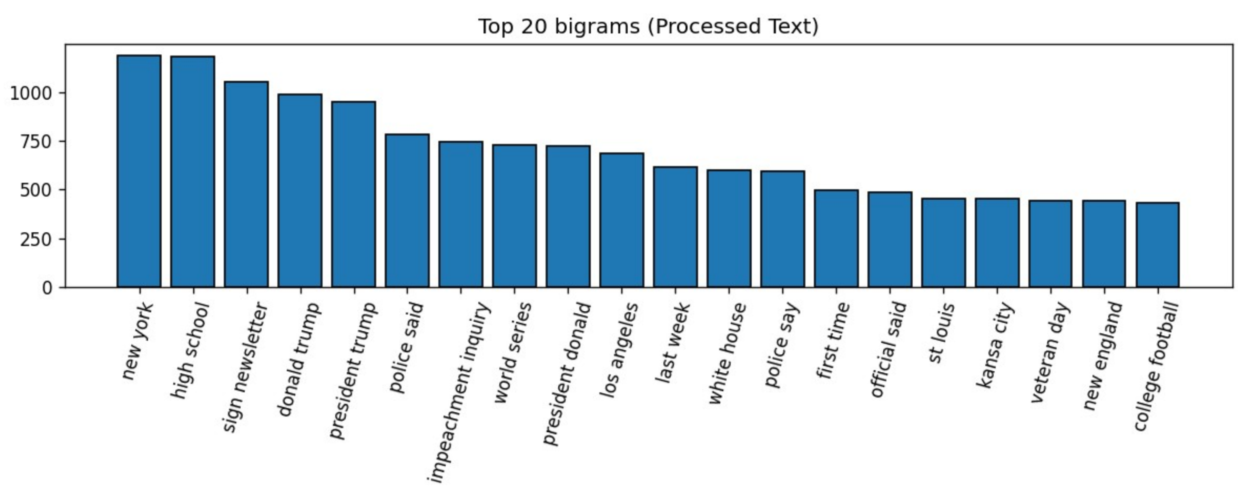
```python
# Length histograms
fig, axes = plt.subplots(1,2, figsize=(12,4))
axes[0].hist(data["len_words"], bins=40, edgecolor="black");
axes[0].set_title("Word Count Distribution")
axes[1].hist(data["len_chars"], bins=40, edgecolor="black");
axes[1].set_title("Character Count Distribution")
plt.tight_layout(); plt.savefig(os.path.join(EXPORTS,
"eda_lengths.png")); plt.show()
```

```
# Word Cloud (subset for speed)
wc = WordCloud(width=1200, height=500,
background_color="white").generate("
".join(data["processed_text"].head(20000)))
plt.figure(figsize=(12,5)); plt.imshow(wc, interpolation="bilinear");
plt.axis("off"); plt.title("Word Cloud (Processed Text)")
plt.savefig(os.path.join(EXPORTS, "wordcloud_processed.png"));
plt.show()

# Top unigrams / bigrams
for n in [1,2]:
    vec = CountVectorizer(ngram_range=(n,n), max_features=30,
min_df=5)
    X = vec.fit_transform(data["processed_text"])
    freqs = np.array(X.sum(axis=0)).ravel()
    vocab = np.array(vec.get_feature_names_out())
    order = np.argsort(-freqs)
    top_vocab = vocab[order][:20]; top_freqs = freqs[order][:20]
    plt.figure(figsize=(10,4))
    plt.bar(range(len(top_vocab)), top_freqs, edgecolor="black")
    plt.xticks(range(len(top_vocab)), top_vocab, rotation=75)
    plt.title(f"Top {20} {'uni' if n==1 else 'bi'}grams (Processed
Text)")
    fname = f"top_{'uni' if n==1 else 'bi'}grams.png"
    plt.tight_layout(); plt.savefig(os.path.join(EXPORTS, fname));
plt.show()
```




Word Cloud (Processed Text)
new
one
said
say
home
week
game
team
make
time
day
first
season
man
two
year
trump
report
according
sunday
state
back
woman
car
take
best
win
way
may
still
three
much
another
family
former
around
people
city
found
world series
donald trump
new york
florida
player
company
show


Top 20 unigrams (Processed Text)
8000
7000
6000
5000
4000
3000
2000
1000
0
new
said
week
year
police
say
one
state
game
first
trump
home
two
time
day
school
county
city
man
season


Top 20 bigrams (Processed Text)
1000
750
500
250
0
new york
high school
sign newsletter
donald trump
president trump
police said
impeachment inquiry
world series
president donald
los angeles
last week
white house
police say
first time
official said
st louis
kansa city
veteran day
new england
college football

Utilities (safe text, chunking, batching, helpers)

```
def safe_text(s: str) -> str:
    s = ("" if s is None else str(s)).strip()
    return s if len(s) >= 5 else ""

def chunks_of(df: pd.DataFrame, chunk_size: int):
    for start in range(0, len(df), chunk_size):
        yield start // chunk_size,
df.iloc[start:start+chunk_size].copy()

def batched(lst, batch_size):
    for i in range(0, len(lst), batch_size):
        yield lst[i:i+batch_size]

def compute_unified(df, weights):
    w = np.array([weights["sent"], weights["emot"], weights["bias"],
weights["cred"]], dtype=float)
    w = w / w.sum()
    M =
df[["sent_norm","emot_norm","bias_norm","cred_norm"]].to_numpy(dtype=f
loat)
    return (M @ w).astype(float)
```

##Pipelines

```
# Global pipelines (loaded on first use)
_sentiment_pipe = None
_emotion_pipe = None
_cred_pipe = None
_bias_pipe = None

def get_sentiment_pipe():
    """
    Use the '-latest' model for human-readable labels + robust mapping
(we also handle LABEL_0/1/2).
    """
    global _sentiment_pipe
    if _sentiment_pipe is None:
        _sentiment_pipe = pipeline(
            "text-classification",
            model="cardiffnlp/twitter-roberta-base-sentiment-latest",
            tokenizer="cardiffnlp/twitter-roberta-base-sentiment-
latest",
            return_all_scores=True,
            truncation=True,
            max_length=CONFIG["max_length"],
            device=DEVICE
        )
    return _sentiment_pipe
```

```python
def get_emotion_pipe():
    global _emotion_pipe
    if _emotion_pipe is None:
        _emotion_pipe = pipeline(
            "text-classification",
            model="j-hartmann/emotion-english-distilroberta-base",
            return_all_scores=True,
            truncation=True,
            max_length=CONFIG["max_length"],
            device=DEVICE
        )
    return _emotion_pipe

def get_cred_pipe():
    global _cred_pipe
    if _cred_pipe is None:
        _cred_pipe = pipeline(
            "text-classification",
            model="jy46604790/Fake-News-Bert-Detect",
            tokenizer="jy46604790/Fake-News-Bert-Detect",
            return_all_scores=True,
            truncation=True,
            max_length=CONFIG["max_length"],
            device=DEVICE
        )
    return _cred_pipe

def get_bias_pipe():
    global _bias_pipe
    if _bias_pipe is None:
        _bias_pipe = pipeline(
            "zero-shot-classification",
            model="facebook/bart-large-mnli",
            device=DEVICE
        )
    return _bias_pipe
```

Mapping Helpers

```python
# --- Sentiment mapping (robust to 'negative/neutral/positive' and
'LABEL_0/1/2') ---
def map_sentiment(scores_list):
    tmp = []
    for d in scores_list:
        raw = d["label"].strip().lower()
        p = float(d["score"])
        if raw in {"label_0","0","neg","negative"}:
            norm = "negative"
```

```
        elif raw in {"label_1","1","neu","neutral"}:
            norm = "neutral"
        elif raw in {"label_2","2","pos","positive"}:
            norm = "positive"
        else:
            if "neg" in raw: norm = "negative"
            elif "neu" in raw: norm = "neutral"
            elif "pos" in raw: norm = "positive"
            else: norm = raw
        tmp.append((norm, p))
    lbl2p = {}
    for lab, p in tmp:
        if lab in {"negative","neutral","positive"}:
            lbl2p[lab] = max(lbl2p.get(lab, 0.0), p)

    p_neg = lbl2p.get("negative",0.0)
    p_neu = lbl2p.get("neutral",0.0)
    p_pos = lbl2p.get("positive",0.0)
    raw_label, conf = max([("Positive",p_pos),("Neutral",p_neu),
("Negative",p_neg)], key=lambda x:x[1])

    if raw_label == "Positive": sent_norm = conf
    elif raw_label == "Neutral": sent_norm = 0.5
    else: sent_norm = 1.0 - conf
    return raw_label, conf, sent_norm

# --- Emotion grouping (GoEmotions → Joy/Sadness/Anger/Fear/Neutral)
---
EMO_GROUPS = {"Joy":{"joy"}, "Sadness":{"sadness"}, "Anger":
{"anger","disgust"}, "Fear":{"fear","surprise"}}

def map_emotion(scores_list):
    lbl2p = {d["label"].strip().lower(): float(d["score"]) for d in
scores_list}
    known = set(lbl2p.keys())
    grouped =
{"Joy":0.0,"Sadness":0.0,"Anger":0.0,"Fear":0.0,"Neutral":0.0}
    for lab in EMO_GROUPS["Joy"]:
        if lab in known: grouped["Joy"] += lbl2p[lab]
    for lab in EMO_GROUPS["Sadness"]:
        if lab in known: grouped["Sadness"] += lbl2p[lab]
    for lab in EMO_GROUPS["Anger"]:
        if lab in known: grouped["Anger"] += lbl2p[lab]
    for lab in EMO_GROUPS["Fear"]:
        if lab in known: grouped["Fear"] += lbl2p[lab]
    explicit = set().union(*EMO_GROUPS.values())
    for lab in known - explicit:
        grouped["Neutral"] += lbl2p[lab]
    emot_label, emot_conf = max(grouped.items(), key=lambda x:x[1])
    if emot_label == "Joy": emot_norm = emot_conf
```

```
    elif emot_label == "Neutral": emot_norm = 0.6
    else: emot_norm = 1.0 - emot_conf
    return emot_label, emot_conf, emot_norm

# --- Bias (zero-shot: Left/Center/Right → Biased/Neutral) ---
BIAS_LABELS = ["Left","Center","Right"]
HYPOTHESIS = "This text is written with a {} political leaning."

def map_bias(zs_output):
    lbl2p = {lab: float(score) for lab, score in
zip(zs_output["labels"], zs_output["scores"])}
    p_left = lbl2p.get("Left",0.0); p_center =
lbl2p.get("Center",0.0); p_right = lbl2p.get("Right",0.0)
    raw_label = max([("Left",p_left),("Right",p_right),
("Center",p_center)], key=lambda x:x[1])[0]
    if raw_label in {"Left","Right"}:
        simp_label = "Biased"; simp_conf = max(p_left,p_right);
bias_norm = 1.0 - simp_conf
    else:
        simp_label = "Neutral"; simp_conf = p_center; bias_norm =
simp_conf
    return simp_label, simp_conf, bias_norm

# --- Credibility (Fake/Real → Low/High) ---
def map_cred(scores_list):
    lbl2p = {d["label"].strip().upper(): float(d["score"]) for d in
scores_list}
    p_fake = max(lbl2p.get("FAKE",0.0), lbl2p.get("LABEL_0",0.0))
    p_real = max(lbl2p.get("REAL",0.0), lbl2p.get("LABEL_1",0.0))
    if p_real >= p_fake:
        label, conf, cred_norm = "High Credibility", p_real, p_real
    else:
        label, conf, cred_norm = "Low Credibility", p_fake, 1.0 -
p_fake
    return label, conf, cred_norm
```

Inference on One Chunk (batched, all four models)

```
def run_models_on_chunk(df_chunk: pd.DataFrame, idx: int) ->
pd.DataFrame:
    texts_raw = df_chunk["combined_text"].tolist()
    texts_proc = df_chunk["processed_text"].tolist()
    texts = texts_proc if CONFIG["use_processed_text"] else texts_raw
    texts = [safe_text(t) for t in texts]

    # --- Sentiment ---
    sent_labels, sent_confs, sent_norms = [], [], []
    spipe = get_sentiment_pipe()
    for batch in tqdm(list(batched(texts, CONFIG["batch_sizes"]
```

```
["sentiment"])), desc=f"[Chunk {idx}] Sentiment"):
        outs = spipe(batch)  # list[list[dict]]
        for scores_list in outs:
            lab, conf, norm = map_sentiment(scores_list)
            sent_labels.append(lab); sent_confs.append(conf);
sent_norms.append(norm)

    # --- Emotion ---
    emot_labels, emot_confs, emot_norms = [], [], []
    epipe = get_emotion_pipe()
    for batch in tqdm(list(batched(texts, CONFIG["batch_sizes"]
["emotion"])), desc=f"[Chunk {idx}] Emotion"):
        outs = epipe(batch)
        for scores_list in outs:
            lab, conf, norm = map_emotion(scores_list)
            emot_labels.append(lab); emot_confs.append(conf);
emot_norms.append(norm)

    # --- Bias (zero-shot) ---
    bias_labels, bias_confs, bias_norms = [], [], []
    bpipe = get_bias_pipe()
    for batch in tqdm(list(batched(texts, CONFIG["batch_sizes"]
["bias"])), desc=f"[Chunk {idx}] Bias ZS"):
        zs_out = bpipe(batch, candidate_labels=BIAS_LABELS,
hypothesis_template=HYPOTHESIS)
        for out in zs_out:
            lab, conf, norm = map_bias(out)
            bias_labels.append(lab); bias_confs.append(conf);
bias_norms.append(norm)

    # --- Credibility ---
    cred_labels, cred_confs, cred_norms = [], [], []
    cpipe = get_cred_pipe()
    for batch in tqdm(list(batched(texts, CONFIG["batch_sizes"]
["cred"])), desc=f"[Chunk {idx}] Cred"):
        outs = cpipe(batch)
        for scores_list in outs:
            lab, conf, norm = map_cred(scores_list)
            cred_labels.append(lab); cred_confs.append(conf);
cred_norms.append(norm)

    out = df_chunk.copy()
    out["sent_label"] = sent_labels; out["sent_conf"] = sent_confs;
out["sent_norm"] = sent_norms
    out["emot_label"] = emot_labels; out["emot_conf"] = emot_confs;
out["emot_norm"] = emot_norms
    out["bias_label"] = bias_labels; out["bias_conf"] = bias_confs;
out["bias_norm"] = bias_norms
    out["cred_label"] = cred_labels; out["cred_conf"] = cred_confs;
```

```
out["cred_norm"] = cred_norms
    return out
```

Run All Chunks (resume-safe, saves per-chunk)

```
CHUNK_SIZE = CONFIG["chunk_size"]
N = len(data); num_chunks = math.ceil(N / CHUNK_SIZE)
print(f"Total rows: {N:,} | Chunks: {num_chunks} | Chunk size:
{CHUNK_SIZE}")

for idx, df_chunk in chunks_of(data, CHUNK_SIZE):
    part_path = os.path.join(RUN_DIR,
f"scores_part_{idx:02d}.parquet")
    if os.path.exists(part_path):
        print(f"Skipping chunk {idx} (already exists).")
        continue
    print(f"Processing chunk {idx} ({len(df_chunk)} rows) ...")
    out_df = run_models_on_chunk(df_chunk, idx)
    out_df.to_parquet(part_path, index=False)
    print("Saved:", part_path)
    del out_df; gc.collect()
```

```
Total rows: 51,282 | Chunks: 11 | Chunk size: 5000
Processing chunk 0 (5000 rows) ...
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/
_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
```


{"model_id":"5d79472ce89a491093c7bf0842dae84a","version_major":2,"vers
ion_minor":0}

{"model_id":"76d9142249574bf0924f4e59701a3588","version_major":2,"vers
ion_minor":0}


```
Some weights of the model checkpoint at cardiffnlp/twitter-roberta-
base-sentiment-latest were not used when initializing
RobertaForSequenceClassification: ['roberta.pooler.dense.bias',
'roberta.pooler.dense.weight']
- This IS expected if you are initializing
RobertaForSequenceClassification from the checkpoint of a model
trained on another task or with another architecture (e.g.
initializing a BertForSequenceClassification model from a
```

```
BertForPreTraining model).
- This IS NOT expected if you are initializing
RobertaForSequenceClassification from the checkpoint of a model that
you expect to be exactly identical (initializing a
BertForSequenceClassification model from a
BertForSequenceClassification model).
```


```
{"model_id":"c5abc9764d9a4cebbda7434c78543f66","version_major":2,"vers
ion_minor":0}

{"model_id":"027a933e3d4f432793cde9193f8eb9d2","version_major":2,"vers
ion_minor":0}

{"model_id":"782262e1a8364e3f8a66bd6cc4b96265","version_major":2,"vers
ion_minor":0}

{"model_id":"0c9ea8c639364f26acec0411363ca683","version_major":2,"vers
ion_minor":0}
```


```
Device set to use cuda:0
/usr/local/lib/python3.12/dist-packages/transformers/pipelines/text_cl
assification.py:111: UserWarning: `return_all_scores` is now
deprecated,  if want a similar functionality use `top_k=None` instead
of `return_all_scores=True` or `top_k=1` instead of
`return_all_scores=False`.
  warnings.warn(

[Chunk 0] Sentiment:   0%|          | 0/157 [00:00<?, ?it/s]ent:   1%|
| 1/157 [00:03<07:48,  3.01s/it]ent:   1%|▏         | 2/157
[00:04<06:01,  2.33s/it]ent:   2%|▏         | 3/157 [00:06<04:50,
1.88s/it]ent:   3%|▎         | 4/157 [00:07<03:57,  1.55s/it]ent:
3%|▎         | 5/157 [00:07<03:00,  1.19s/it]ent:   4%|▍         |
6/157 [00:08<02:12,  1.14it/s]ent:   4%|▍         | 7/157
[00:08<01:41,  1.48it/s]ent:   5%|▌         | 8/157 [00:08<01:21,
1.84it/s]ent:   6%|▌         | 9/157 [00:08<01:07,  2.18it/s]ent:
6%|▋         | 10/157 [00:09<00:59,  2.49it/s] to be using the
pipelines sequentially on GPU. In order to maximize efficiency please
use a dataset

[Chunk 0] Sentiment:   7%|▋         | 11/157 [00:09<00:53,
2.74it/s]ent:   8%|▊         | 12/157 [00:09<00:51,  2.81it/s]ent:
8%|▊         | 13/157 [00:10<00:53,  2.70it/s]ent:   9%|▉         |
14/157 [00:10<00:54,  2.64it/s]ent:  10%|▉         | 15/157
[00:10<00:50,  2.80it/s]ent:  10%|█         | 16/157 [00:11<00:46,
3.03it/s]ent:  11%|█         | 17/157 [00:11<00:46,  3.01it/s]ent:
11%|█▏        | 18/157 [00:11<00:45,  3.08it/s]ent:  12%|█▏        |
19/157 [00:12<00:46,  2.95it/s]ent:  13%|█▎        | 20/157
[00:12<00:46,  2.95it/s]ent:  13%|█▎        | 21/157 [00:12<00:48,
2.82it/s]ent:  14%|█▍        | 22/157 [00:13<00:55,  2.44it/s]ent:
15%|█▍        | 23/157 [00:13<00:56,  2.36it/s]ent:  15%|█▌        |
24/157 [00:14<00:52,  2.52it/s]ent:  16%|█▌        | 25/157
```

```
[00:14<00:47,  2.75it/s]ent:  17%|█▋        | 26/157 [00:14<00:51,
2.55it/s]ent:  17%|█▋        | 27/157 [00:15<00:48,  2.66it/s]ent:
18%|█▊        | 28/157 [00:15<00:46,  2.75it/s]ent:  18%|█▊        |
29/157 [00:15<00:45,  2.80it/s]ent:  19%|█▉        | 30/157
[00:16<00:42,  2.98it/s]ent:  20%|█▉        | 31/157 [00:16<00:40,
3.10it/s]ent:  20%|██        | 32/157 [00:16<00:38,  3.28it/s]ent:
21%|██        | 33/157 [00:17<00:36,  3.39it/s]ent:  22%|██▏       |
34/157 [00:17<00:38,  3.15it/s]ent:  22%|██▏       | 35/157
[00:17<00:40,  3.04it/s]ent:  23%|██▎       | 36/157 [00:18<00:40,
3.00it/s]ent:  24%|██▎       | 37/157 [00:18<00:40,  2.99it/s]ent:
24%|██▍       | 38/157 [00:18<00:41,  2.90it/s]ent:  25%|██▍       |
39/157 [00:19<00:40,  2.91it/s]ent:  25%|██▌       | 40/157
[00:19<00:40,  2.90it/s]ent:  26%|██▌       | 41/157 [00:19<00:42,
2.74it/s]ent:  27%|██▋       | 42/157 [00:20<00:44,  2.59it/s]ent:
27%|██▋       | 43/157 [00:20<00:43,  2.63it/s]ent:  28%|██▊       |
44/157 [00:21<00:39,  2.85it/s]ent:  29%|██▊       | 45/157
[00:21<00:35,  3.12it/s]ent:  29%|██▉       | 46/157 [00:21<00:33,
3.34it/s]ent:  30%|██▉       | 47/157 [00:21<00:32,  3.38it/s]ent:
31%|███       | 48/157 [00:22<00:31,  3.50it/s]ent:  31%|███       |
49/157 [00:22<00:29,  3.63it/s]ent:  32%|███▏      | 50/157
[00:22<00:28,  3.72it/s]ent:  32%|███▏      | 51/157 [00:22<00:28,
3.68it/s]ent:  33%|███▎      | 52/157 [00:23<00:28,  3.66it/s]ent:
34%|███▍      | 53/157 [00:23<00:27,  3.74it/s]ent:  34%|███▍      |
54/157 [00:23<00:27,  3.77it/s]ent:  35%|███▌      | 55/157
[00:23<00:27,  3.68it/s]ent:  36%|███▌      | 56/157 [00:24<00:26,
3.75it/s]ent:  36%|███▋      | 57/157 [00:24<00:26,  3.82it/s]ent:
37%|███▋      | 58/157 [00:24<00:25,  3.83it/s]ent:  38%|███▊      |
59/157 [00:25<00:26,  3.70it/s]ent:  38%|███▊      | 60/157
[00:25<00:25,  3.77it/s]ent:  39%|███▉      | 61/157 [00:25<00:25,
3.80it/s]ent:  39%|███▉      | 62/157 [00:25<00:24,  3.81it/s]ent:
40%|████      | 63/157 [00:26<00:25,  3.70it/s]ent:  41%|████      |
64/157 [00:26<00:24,  3.77it/s]ent:  41%|████▏     | 65/157
[00:26<00:24,  3.79it/s]ent:  42%|████▏     | 66/157 [00:26<00:23,
3.81it/s]ent:  43%|████▎     | 67/157 [00:27<00:24,  3.73it/s]ent:
43%|████▎     | 68/157 [00:27<00:23,  3.75it/s]ent:  44%|████▍     |
69/157 [00:27<00:23,  3.76it/s]ent:  45%|████▍     | 70/157
[00:27<00:23,  3.71it/s]ent:  45%|████▌     | 71/157 [00:28<00:22,
3.74it/s]ent:  46%|████▌     | 72/157 [00:28<00:22,  3.79it/s]ent:
46%|████▋     | 73/157 [00:28<00:22,  3.78it/s]ent:  47%|████▋     |
74/157 [00:28<00:22,  3.71it/s]ent:  48%|████▊     | 75/157
[00:29<00:21,  3.74it/s]ent:  48%|████▊     | 76/157 [00:29<00:21,
3.78it/s]ent:  49%|████▉     | 77/157 [00:29<00:21,  3.79it/s]ent:
50%|████▉     | 78/157 [00:30<00:21,  3.70it/s]ent:  50%|█████     |
79/157 [00:30<00:20,  3.77it/s]ent:  51%|█████     | 80/157
[00:30<00:21,  3.53it/s]ent:  52%|█████▏    | 81/157 [00:31<00:23,
3.19it/s]ent:  52%|█████▏    | 82/157 [00:31<00:24,  3.04it/s]ent:
53%|█████▎    | 83/157 [00:31<00:24,  2.97it/s]ent:  54%|█████▎    |
84/157 [00:32<00:25,  2.90it/s]ent:  54%|█████▍    | 85/157
[00:32<00:25,  2.81it/s]ent:  55%|█████▍    | 86/157 [00:32<00:25,
```

```
2.79it/s]ent:  55%|█████     | 87/157 [00:33<00:26,  2.63it/s]ent:
56%|█████     | 88/157 [00:33<00:27,  2.51it/s]ent:  57%|█████     |
89/157 [00:34<00:25,  2.65it/s]ent:  57%|█████     | 90/157
[00:34<00:23,  2.87it/s]ent:  58%|█████     | 91/157 [00:34<00:21,
3.10it/s]ent:  59%|█████     | 92/157 [00:34<00:20,  3.21it/s]ent:
59%|█████     | 93/157 [00:35<00:19,  3.33it/s]ent:  60%|██████    |
94/157 [00:35<00:18,  3.40it/s]ent:  61%|██████    | 95/157
[00:35<00:17,  3.50it/s]ent:  61%|██████    | 96/157 [00:35<00:17,
3.54it/s]ent:  62%|██████    | 97/157 [00:36<00:16,  3.60it/s]ent:
62%|██████    | 98/157 [00:36<00:16,  3.58it/s]ent:  63%|██████    |
99/157 [00:36<00:16,  3.57it/s]ent:  64%|██████    | 100/157
[00:37<00:15,  3.59it/s]ent:  64%|██████    | 101/157 [00:37<00:15,
3.64it/s]ent:  65%|██████    | 102/157 [00:37<00:15,  3.58it/s]ent:
66%|██████    | 103/157 [00:37<00:14,  3.64it/s]ent:  66%|██████    |
104/157 [00:38<00:14,  3.67it/s]ent:  67%|██████    | 105/157
[00:38<00:14,  3.70it/s]ent:  68%|██████    | 106/157 [00:38<00:13,
3.65it/s]ent:  68%|██████    | 107/157 [00:38<00:13,  3.69it/s]ent:
69%|██████    | 108/157 [00:39<00:13,  3.71it/s]ent:  69%|██████    |
109/157 [00:39<00:13,  3.68it/s]ent:  70%|███████   | 110/157
[00:39<00:12,  3.68it/s]ent:  71%|███████   | 111/157 [00:40<00:12,
3.71it/s]ent:  71%|███████   | 112/157 [00:40<00:12,  3.73it/s]ent:
72%|███████   | 113/157 [00:40<00:11,  3.68it/s]ent:  73%|███████   |
114/157 [00:40<00:11,  3.66it/s]ent:  73%|███████   | 115/157
[00:41<00:11,  3.62it/s]ent:  74%|███████   | 116/157 [00:41<00:11,
3.68it/s]ent:  75%|███████   | 117/157 [00:41<00:10,  3.65it/s]ent:
75%|███████   | 118/157 [00:41<00:10,  3.69it/s]ent:  76%|███████   |
119/157 [00:42<00:10,  3.64it/s]ent:  76%|███████   | 120/157
[00:42<00:10,  3.63it/s]ent:  77%|███████   | 121/157 [00:42<00:10,
3.57it/s]ent:  78%|███████   | 122/157 [00:43<00:09,  3.63it/s]ent:
78%|███████   | 123/157 [00:43<00:09,  3.70it/s]ent:  79%|███████   |
124/157 [00:43<00:08,  3.73it/s]ent:  80%|████████  | 125/157
[00:43<00:08,  3.57it/s]ent:  80%|████████  | 126/157 [00:44<00:09,
3.22it/s]ent:  81%|████████  | 127/157 [00:44<00:09,  3.09it/s]ent:
82%|████████  | 128/157 [00:45<00:09,  2.95it/s]ent:  82%|████████  |
129/157 [00:45<00:09,  2.92it/s]ent:  83%|████████  | 130/157
[00:45<00:09,  2.88it/s]ent:  83%|████████  | 131/157 [00:46<00:09,
2.83it/s]ent:  84%|████████  | 132/157 [00:46<00:09,  2.68it/s]ent:
85%|████████  | 133/157 [00:46<00:09,  2.59it/s]ent:  85%|████████  |
134/157 [00:47<00:08,  2.63it/s]ent:  86%|████████  | 135/157
[00:47<00:07,  2.83it/s]ent:  87%|████████  | 136/157 [00:47<00:06,
3.06it/s]ent:  87%|████████  | 137/157 [00:48<00:06,  3.18it/s]ent:
88%|████████  | 138/157 [00:48<00:05,  3.35it/s]ent:  89%|████████  |
139/157 [00:48<00:05,  3.45it/s]ent:  89%|████████  | 140/157
[00:48<00:04,  3.51it/s]ent:  90%|█████████ | 141/157 [00:49<00:04,
3.50it/s]ent:  90%|█████████ | 142/157 [00:49<00:04,  3.55it/s]ent:
91%|█████████ | 143/157 [00:49<00:03,  3.60it/s]ent:  92%|█████████ |
144/157 [00:50<00:03,  3.57it/s]ent:  92%|█████████ | 145/157
[00:50<00:03,  3.66it/s]ent:  93%|█████████ | 146/157 [00:50<00:02,
3.70it/s]ent:  94%|█████████ | 147/157 [00:50<00:02,  3.61it/s]ent:
```

```
94%|█████████▍| 148/157 [00:51<00:02,  3.61it/s]ent:  95%|█████████▍| 149/157 [00:51<00:02,  3.72it/s]ent:  96%|█████████▌| 150/157 [00:51<00:01,  3.78it/s]ent:  96%|█████████▌| 151/157 [00:51<00:01,  3.81it/s]ent:  97%|█████████▋| 152/157 [00:52<00:01,  3.76it/s]ent:  97%|█████████▋| 153/157 [00:52<00:01,  3.81it/s]ent:  98%|█████████▊| 154/157 [00:52<00:00,  3.81it/s]ent:  99%|█████████▊| 155/157 [00:52<00:00,  3.84it/s]ent: 100%|██████████| 157/157 [00:53<00:00,  2.94it/s]
```

{"model_id":"e1f10a8efb814d548a6b776f8c26ecd5","version_major":2,"version_minor":0}

{"model_id":"07966e153c3d4cf78866dc91d79eb54a","version_major":2,"version_minor":0}

{"model_id":"2948f0a916bb40c4b759e58513c93e0a","version_major":2,"version_minor":0}

{"model_id":"17825fc4118d4260806cd91cccc35bb4","version_major":2,"version_minor":0}

{"model_id":"c4a8b66067b1410992a502f18b3865b8","version_major":2,"version_minor":0}

{"model_id":"0ccb9bc81bf144fe990bf6684eb0d489","version_major":2,"version_minor":0}

{"model_id":"7da6218a1e29467b8b1e75ee0e10151d","version_major":2,"version_minor":0}

{"model_id":"93bde292e82145a187985e1bd55ce081","version_major":2,"version_minor":0}

```
Device set to use cuda:0
```

```
[Chunk 0] Emotion:   0%|          | 0/209 [00:00<?, ?it/s]otion:   0%|          | 1/209 [00:00<02:38,  1.32it/s]otion:   1%|          | 2/209 [00:01<02:45,  1.25it/s]otion:   1%|▏         | 3/209 [00:02<02:58,  1.15it/s]otion:   2%|▏         | 4/209 [00:03<03:02,  1.12it/s]otion:   2%|▏         | 5/209 [00:03<02:07,  1.60it/s]otion:   3%|▎         | 6/209 [00:03<01:32,  2.19it/s]otion:   3%|▎         | 7/209 [00:03<01:10,  2.87it/s]otion:   4%|▍         | 8/209 [00:04<00:56,  3.54it/s]otion:   4%|▍         | 9/209 [00:04<00:48,  4.13it/s]otion:   5%|▍         | 10/209 [00:04<00:41,  4.83it/s]otion:   5%|▌         | 11/209 [00:04<00:37,  5.22it/s]otion:   6%|▌         | 12/209 [00:04<00:36,  5.39it/s]otion:   6%|▌         | 13/209 [00:04<00:35,  5.47it/s]otion:   7%|▋         | 14/209 [00:04<00:35,  5.57it/s]otion:   7%|▋         | 15/209 [00:05<00:34,  5.55it/s]otion:   8%|▊         | 16/209 [00:05<00:34,  5.67it/s]otion:   8%|▊         | 17/209 [00:05<00:32,  5.83it/s]otion:   9%|▊         | 18/209 [00:05<00:42,  4.54it/s]otion:   9%|▉         | 19/209 [00:06<00:52,  3.62it/s]otion:
```

```
10%|▉         | 20/209 [00:06<01:01,  3.08it/s]otion:  10%|█         |
21/209 [00:06<00:57,  3.29it/s]otion:  11%|█         | 22/209
[00:07<00:48,  3.85it/s]otion:  11%|█         | 23/209 [00:07<00:43,
4.24it/s]otion:  11%|█▏        | 24/209 [00:07<00:41,  4.49it/s]otion:
12%|█▏        | 25/209 [00:07<00:38,  4.77it/s]otion:  12%|█▏        |
26/209 [00:07<00:36,  5.01it/s]otion:  13%|█▎        | 27/209
[00:07<00:36,  5.03it/s]otion:  13%|█▎        | 28/209 [00:08<00:35,
5.10it/s]otion:  14%|█▍        | 29/209 [00:08<00:33,  5.40it/s]otion:
14%|█▍        | 30/209 [00:08<00:29,  6.05it/s]otion:  15%|█▍        |
31/209 [00:08<00:31,  5.65it/s]otion:  15%|█▌        | 32/209
[00:08<00:32,  5.37it/s]otion:  16%|█▌        | 33/209 [00:09<00:29,
5.97it/s]otion:  16%|█▋        | 34/209 [00:09<00:27,  6.46it/s]otion:
17%|█▋        | 35/209 [00:09<00:25,  6.96it/s]otion:  17%|█▋        |
36/209 [00:09<00:24,  7.09it/s]otion:  18%|█▊        | 37/209
[00:09<00:25,  6.88it/s]otion:  18%|█▊        | 38/209 [00:09<00:30,
5.64it/s]otion:  19%|█▊        | 39/209 [00:09<00:28,  5.97it/s]otion:
19%|█▉        | 40/209 [00:10<00:29,  5.82it/s]otion:  20%|█▉        |
41/209 [00:10<00:33,  5.01it/s]otion:  20%|██        | 42/209
[00:10<00:29,  5.61it/s]otion:  21%|██        | 43/209 [00:10<00:26,
6.26it/s]otion:  21%|██        | 44/209 [00:10<00:26,  6.20it/s]otion:
22%|██▏       | 45/209 [00:10<00:25,  6.37it/s]otion:  22%|██▏       |
46/209 [00:11<00:23,  6.86it/s]otion:  22%|██▏       | 47/209
[00:11<00:23,  7.02it/s]otion:  23%|██▎       | 48/209 [00:11<00:23,
6.83it/s]otion:  23%|██▎       | 49/209 [00:11<00:25,  6.19it/s]otion:
24%|██▍       | 50/209 [00:11<00:29,  5.34it/s]otion:  24%|██▍       |
51/209 [00:11<00:29,  5.40it/s]otion:  25%|██▍       | 52/209
[00:12<00:29,  5.31it/s]otion:  25%|██▌       | 53/209 [00:12<00:27,
5.58it/s]otion:  26%|██▌       | 54/209 [00:12<00:26,  5.96it/s]otion:
26%|██▋       | 55/209 [00:12<00:23,  6.51it/s]otion:  27%|██▋       |
56/209 [00:12<00:21,  7.00it/s]otion:  27%|██▋       | 57/209
[00:12<00:20,  7.30it/s]otion:  28%|██▊       | 58/209 [00:12<00:19,
7.63it/s]otion:  28%|██▊       | 59/209 [00:13<00:19,  7.59it/s]otion:
29%|██▊       | 60/209 [00:13<00:21,  6.81it/s]otion:  29%|██▉       |
61/209 [00:13<00:20,  7.06it/s]otion:  30%|██▉       | 62/209
[00:13<00:20,  7.04it/s]otion:  30%|███       | 63/209 [00:13<00:24,
5.85it/s]otion:  31%|███       | 64/209 [00:13<00:24,  5.84it/s]otion:
31%|███       | 65/209 [00:14<00:22,  6.27it/s]otion:  32%|███▏      |
66/209 [00:14<00:21,  6.76it/s]otion:  32%|███▏      | 67/209
[00:14<00:19,  7.24it/s]otion:  33%|███▎      | 68/209 [00:14<00:18,
7.51it/s]otion:  33%|███▎      | 69/209 [00:14<00:18,  7.55it/s]otion:
33%|███▎      | 70/209 [00:14<00:17,  7.85it/s]otion:  34%|███▍      |
71/209 [00:14<00:17,  7.96it/s]otion:  34%|███▍      | 72/209
[00:14<00:16,  8.12it/s]otion:  35%|███▍      | 73/209 [00:15<00:16,
8.20it/s]otion:  35%|███▌      | 74/209 [00:15<00:16,  8.17it/s]otion:
36%|███▌      | 75/209 [00:15<00:16,  8.22it/s]otion:  36%|███▋      |
76/209 [00:15<00:16,  8.25it/s]otion:  37%|███▋      | 77/209
[00:15<00:15,  8.27it/s]otion:  37%|███▋      | 78/209 [00:15<00:16,
7.98it/s]otion:  38%|███▊      | 79/209 [00:15<00:16,  8.03it/s]otion:
38%|███▊      | 80/209 [00:15<00:16,  7.69it/s]otion:  39%|███▉      |
```

```
81/209 [00:16<00:17,  7.43it/s]otion:  39%|███▉      | 82/209
[00:16<00:16,  7.67it/s]otion:  40%|███▉      | 83/209 [00:16<00:15,
7.88it/s]otion:  40%|████      | 84/209 [00:16<00:15,  8.02it/s]otion:
41%|████      | 85/209 [00:16<00:15,  8.18it/s]otion:  41%|████      |
86/209 [00:16<00:15,  7.89it/s]otion:  42%|████▏     | 87/209
[00:16<00:15,  7.86it/s]otion:  42%|████▏     | 88/209 [00:16<00:15,
7.86it/s]otion:  43%|████▎     | 89/209 [00:17<00:15,  7.88it/s]otion:
43%|████▎     | 90/209 [00:17<00:14,  7.95it/s]otion:  44%|████▎     |
91/209 [00:17<00:14,  8.05it/s]otion:  44%|████▍     | 92/209
[00:17<00:14,  8.04it/s]otion:  44%|████▍     | 93/209 [00:17<00:14,
8.07it/s]otion:  45%|████▍     | 94/209 [00:17<00:14,  7.81it/s]otion:
45%|████▌     | 95/209 [00:17<00:14,  7.91it/s]otion:  46%|████▌     |
96/209 [00:17<00:14,  8.01it/s]otion:  46%|████▋     | 97/209
[00:18<00:13,  8.01it/s]otion:  47%|████▋     | 98/209 [00:18<00:13,
7.97it/s]otion:  47%|████▋     | 99/209 [00:18<00:14,  7.45it/s]otion:
48%|████▊     | 100/209 [00:18<00:15,  6.83it/s]otion:  48%|████▊
| 101/209 [00:18<00:16,  6.53it/s]otion:  49%|████▉     | 102/209
[00:18<00:16,  6.32it/s]otion:  49%|████▉     | 103/209 [00:19<00:16,
6.35it/s]otion:  50%|████▉     | 104/209 [00:19<00:16,
6.35it/s]otion:  50%|█████     | 105/209 [00:19<00:16,
6.36it/s]otion:  51%|█████     | 106/209 [00:19<00:15,
6.44it/s]otion:  51%|█████     | 107/209 [00:19<00:15,
6.52it/s]otion:  52%|█████▏    | 108/209 [00:19<00:16,
6.26it/s]otion:  52%|█████▏    | 109/209 [00:19<00:16,
6.12it/s]otion:  53%|█████▎    | 110/209 [00:20<00:15,
6.20it/s]otion:  53%|█████▎    | 111/209 [00:20<00:15,
6.18it/s]otion:  54%|█████▎    | 112/209 [00:20<00:15,
6.30it/s]otion:  54%|█████▍    | 113/209 [00:20<00:14,
6.41it/s]otion:  55%|█████▍    | 114/209 [00:20<00:15,
6.05it/s]otion:  55%|█████▌    | 115/209 [00:20<00:16,
5.61it/s]otion:  56%|█████▌    | 116/209 [00:21<00:16,
5.63it/s]otion:  56%|█████▌    | 117/209 [00:21<00:16,
5.52it/s]otion:  56%|█████▋    | 118/209 [00:21<00:16,
5.46it/s]otion:  57%|█████▋    | 119/209 [00:21<00:16,
5.59it/s]otion:  57%|█████▋    | 120/209 [00:21<00:14,
6.19it/s]otion:  58%|█████▊    | 121/209 [00:21<00:13,
6.58it/s]otion:  58%|█████▊    | 122/209 [00:22<00:12,
7.05it/s]otion:  59%|█████▉    | 123/209 [00:22<00:11,
7.39it/s]otion:  59%|█████▉    | 124/209 [00:22<00:11,
7.65it/s]otion:  60%|█████▉    | 125/209 [00:22<00:10,
7.86it/s]otion:  60%|██████    | 126/209 [00:22<00:10,
8.12it/s]otion:  61%|██████    | 127/209 [00:22<00:09,
8.26it/s]otion:  61%|██████    | 128/209 [00:22<00:09,
8.24it/s]otion:  62%|██████▏   | 129/209 [00:22<00:09,
8.01it/s]otion:  62%|██████▏   | 130/209 [00:23<00:09,
8.10it/s]otion:  63%|██████▎   | 131/209 [00:23<00:09,
8.16it/s]otion:  63%|██████▎   | 132/209 [00:23<00:09,
8.25it/s]otion:  64%|██████▎   | 133/209 [00:23<00:09,
8.23it/s]otion:  64%|██████▍   | 134/209 [00:23<00:09,
```

```
8.26it/s]otion:  65%|      | 135/209 [00:23<00:09,
8.13it/s]otion:  65%|      | 136/209 [00:23<00:09,
8.06it/s]otion:  66%|      | 137/209 [00:23<00:08,
8.22it/s]otion:  66%|      | 138/209 [00:24<00:08,
7.92it/s]otion:  67%|      | 139/209 [00:24<00:08,
8.05it/s]otion:  67%|      | 140/209 [00:24<00:08,
8.19it/s]otion:  67%|      | 141/209 [00:24<00:08,
8.25it/s]otion:  68%|      | 142/209 [00:24<00:08,
8.29it/s]otion:  68%|      | 143/209 [00:24<00:07,
8.45it/s]otion:  69%|      | 144/209 [00:24<00:07,
8.40it/s]otion:  69%|      | 145/209 [00:24<00:07,
8.33it/s]otion:  70%|      | 146/209 [00:24<00:07,
8.04it/s]otion:  70%|      | 147/209 [00:25<00:07,
8.07it/s]otion:  71%|      | 148/209 [00:25<00:07,
8.17it/s]otion:  71%|      | 149/209 [00:25<00:07,
8.21it/s]otion:  72%|      | 150/209 [00:25<00:07,
8.26it/s]otion:  72%|      | 151/209 [00:25<00:06,
8.30it/s]otion:  73%|      | 152/209 [00:25<00:06,
8.32it/s]otion:  73%|      | 153/209 [00:25<00:06,
8.18it/s]otion:  74%|      | 154/209 [00:25<00:06,
7.93it/s]otion:  74%|      | 155/209 [00:26<00:07,
7.48it/s]otion:  75%|      | 156/209 [00:26<00:07,
7.52it/s]otion:  75%|      | 157/209 [00:26<00:11,
4.37it/s]otion:  76%|      | 158/209 [00:27<00:15,
3.30it/s]otion:  76%|      | 159/209 [00:27<00:18,
2.77it/s]otion:  77%|      | 160/209 [00:28<00:19,
2.58it/s]otion:  77%|      | 161/209 [00:28<00:19,
2.50it/s]otion:  78%|      | 162/209 [00:28<00:17,
2.67it/s]otion:  78%|      | 163/209 [00:29<00:18,
2.43it/s]otion:  78%|      | 164/209 [00:29<00:18,
2.40it/s]otion:  79%|      | 165/209 [00:30<00:20,
2.16it/s]otion:  79%|      | 166/209 [00:30<00:20,
2.13it/s]otion:  80%|      | 167/209 [00:31<00:18,
2.22it/s]otion:  80%|      | 168/209 [00:31<00:17,
2.37it/s]otion:  81%|      | 169/209 [00:32<00:23,
1.72it/s]otion:  81%|      | 170/209 [00:33<00:25,
1.53it/s]otion:  82%|      | 171/209 [00:34<00:26,
1.44it/s]otion:  82%|      | 172/209 [00:34<00:19,
1.87it/s]otion:  83%|      | 173/209 [00:34<00:15,
2.33it/s]otion:  83%|      | 174/209 [00:35<00:18,
1.91it/s]otion:  84%|      | 175/209 [00:36<00:20,
1.70it/s]otion:  84%|      | 176/209 [00:36<00:18,
1.75it/s]otion:  85%|      | 177/209 [00:37<00:17,
1.80it/s]otion:  85%|      | 178/209 [00:37<00:16,
1.90it/s]otion:  86%|      | 179/209 [00:37<00:14,
2.10it/s]otion:  86%|      | 180/209 [00:38<00:13,
2.13it/s]otion:  87%|      | 181/209 [00:38<00:11,
2.38it/s]otion:  87%|      | 182/209 [00:38<00:09,
2.94it/s]otion:  88%|      | 183/209 [00:38<00:07,
```

```
3.60it/s]otion:  88%|█████████  | 184/209 [00:39<00:05,
4.28it/s]otion:  89%|█████████  | 185/209 [00:39<00:04,
4.99it/s]otion:  89%|█████████  | 186/209 [00:39<00:04,
5.67it/s]otion:  89%|█████████  | 187/209 [00:39<00:03,
6.25it/s]otion:  90%|█████████  | 188/209 [00:39<00:03,
6.79it/s]otion:  90%|█████████  | 189/209 [00:39<00:02,
7.22it/s]otion:  91%|█████████  | 190/209 [00:39<00:02,
7.40it/s]otion:  91%|█████████  | 191/209 [00:39<00:02,
7.39it/s]otion:  92%|█████████  | 192/209 [00:40<00:02,
7.64it/s]otion:  92%|█████████  | 193/209 [00:40<00:02,
7.75it/s]otion:  93%|█████████▎ | 194/209 [00:40<00:01,
7.96it/s]otion:  93%|█████████▎ | 195/209 [00:40<00:01,
8.04it/s]otion:  94%|█████████▍ | 196/209 [00:40<00:01,
8.18it/s]otion:  94%|█████████▍ | 197/209 [00:40<00:01,
8.21it/s]otion:  95%|█████████▍ | 198/209 [00:40<00:01,
8.20it/s]otion:  95%|█████████▌ | 199/209 [00:40<00:01,
7.83it/s]otion:  96%|█████████▌ | 200/209 [00:41<00:01,
7.86it/s]otion:  96%|█████████▌ | 201/209 [00:41<00:01,
7.96it/s]otion:  97%|█████████▋ | 202/209 [00:41<00:00,
8.07it/s]otion:  97%|█████████▋ | 203/209 [00:41<00:00,
8.10it/s]otion:  98%|█████████▊ | 204/209 [00:41<00:00,
8.08it/s]otion:  98%|█████████▊ | 205/209 [00:41<00:00,
8.12it/s]otion:  99%|█████████▊ | 206/209 [00:41<00:00,
7.99it/s]otion:  99%|█████████▉ | 207/209 [00:41<00:00,
7.91it/s]otion: 100%|██████████| 209/209 [00:42<00:00,  4.97it/s]
```

{"model_id":"f1f4c3de9e2e48fca98228d3587e0a13","version_major":2,"version_minor":0}

{"model_id":"d0da691aad7f45d5968c37704c52ce31","version_major":2,"version_minor":0}

{"model_id":"0caccd0426224f76af73815a52b05743","version_major":2,"version_minor":0}

{"model_id":"f46c62443f5242188d5e2d32825b1076","version_major":2,"version_minor":0}

{"model_id":"776c9277d46149e59e7276205d490d39","version_major":2,"version_minor":0}

{"model_id":"3d34c6cd36ae428991767de48df7da0b","version_major":2,"version_minor":0}

```
Device set to use cuda:0
[Chunk 0] Bias ZS: 100%|██████████| 834/834 [05:59<00:00,  2.32it/s]
```

{"model_id":"33e7bbd92ed848b5b8af42b30896e24b","version_major":2,"version_minor":0}

{"model_id":"03a33816754341eb9e9ee70d0bec873b","version_major":2,"version_minor":0}

{"model_id":"0f3d72c7fd4b4853a15b23ce31bff3ab","version_major":2,"version_minor":0}

{"model_id":"27a93dacb043430e82e2bddf47652fd7","version_major":2,"version_minor":0}

{"model_id":"8c583c10795448d58642a14b74fcf38a","version_major":2,"version_minor":0}

{"model_id":"84116f7fdba84e8e8d968e31365c06d2","version_major":2,"version_minor":0}

{"model_id":"2e233afce3a545e0ae682bf1a3f06fd4","version_major":2,"version_minor":0}

```
Device set to use cuda:0

[Chunk 0] Cred:   0%|          | 0/209 [00:00<?, ?it/s]

Saved: /content/drive/MyDrive/Final
Project/run_outputs/scores_part_00.parquet
Processing chunk 1 (5000 rows) ...

[Chunk 1] Sentiment: 100%|██████████| 157/157 [00:44<00:00,  3.56it/s]
[Chunk 1] Emotion: 100%|██████████| 209/209 [00:25<00:00,  8.08it/s]
[Chunk 1] Bias ZS: 100%|██████████| 834/834 [05:55<00:00,  2.35it/s]
[Chunk 1] Cred: 100%|██████████| 209/209 [00:42<00:00,  4.87it/s]

Saved: /content/drive/MyDrive/Final
Project/run_outputs/scores_part_01.parquet
Processing chunk 2 (5000 rows) ...

[Chunk 2] Sentiment: 100%|██████████| 157/157 [00:42<00:00,  3.68it/s]
[Chunk 2] Emotion: 100%|██████████| 209/209 [00:26<00:00,  8.03it/s]
[Chunk 2] Bias ZS: 100%|██████████| 834/834 [06:02<00:00,  2.30it/s]
[Chunk 2] Cred: 100%|██████████| 209/209 [00:42<00:00,  4.88it/s]

Saved: /content/drive/MyDrive/Final
Project/run_outputs/scores_part_02.parquet
Processing chunk 3 (5000 rows) ...

[Chunk 3] Sentiment: 100%|██████████| 157/157 [00:43<00:00,  3.63it/s]
[Chunk 3] Emotion: 100%|██████████| 209/209 [00:25<00:00,  8.09it/s]
[Chunk 3] Bias ZS: 100%|██████████| 834/834 [05:59<00:00,  2.32it/s]
[Chunk 3] Cred: 100%|██████████| 209/209 [00:43<00:00,  4.77it/s]

Saved: /content/drive/MyDrive/Final
Project/run_outputs/scores_part_03.parquet
Processing chunk 4 (5000 rows) ...
```

```
[Chunk 4] Sentiment: 100%|██████████| 157/157 [00:44<00:00,  3.52it/s]
[Chunk 4] Emotion: 100%|██████████| 209/209 [00:26<00:00,  7.76it/s]
[Chunk 4] Bias ZS: 100%|██████████| 834/834 [06:03<00:00,  2.29it/s]
[Chunk 4] Cred: 100%|██████████| 209/209 [00:44<00:00,  4.68it/s]

Saved: /content/drive/MyDrive/Final
Project/run_outputs/scores_part_04.parquet
Processing chunk 5 (5000 rows) ...

[Chunk 5] Sentiment: 100%|██████████| 157/157 [00:43<00:00,  3.61it/s]
[Chunk 5] Emotion: 100%|██████████| 209/209 [00:26<00:00,  7.82it/s]
[Chunk 5] Bias ZS: 100%|██████████| 834/834 [06:04<00:00,  2.29it/s]
[Chunk 5] Cred: 100%|██████████| 209/209 [00:43<00:00,  4.81it/s]

Saved: /content/drive/MyDrive/Final
Project/run_outputs/scores_part_05.parquet
Processing chunk 6 (5000 rows) ...

[Chunk 6] Sentiment: 100%|██████████| 157/157 [00:44<00:00,  3.57it/s]
[Chunk 6] Emotion: 100%|██████████| 209/209 [00:26<00:00,  7.93it/s]
[Chunk 6] Bias ZS: 100%|██████████| 834/834 [06:04<00:00,  2.29it/s]
[Chunk 6] Cred: 100%|██████████| 209/209 [00:43<00:00,  4.82it/s]

Saved: /content/drive/MyDrive/Final
Project/run_outputs/scores_part_06.parquet
Processing chunk 7 (5000 rows) ...

[Chunk 7] Sentiment: 100%|██████████| 157/157 [00:42<00:00,  3.66it/s]
[Chunk 7] Emotion: 100%|██████████| 209/209 [00:25<00:00,  8.09it/s]
[Chunk 7] Bias ZS: 100%|██████████| 834/834 [06:10<00:00,  2.25it/s]
[Chunk 7] Cred: 100%|██████████| 209/209 [00:43<00:00,  4.85it/s]

Saved: /content/drive/MyDrive/Final
Project/run_outputs/scores_part_07.parquet
Processing chunk 8 (5000 rows) ...

[Chunk 8] Sentiment: 100%|██████████| 157/157 [00:43<00:00,  3.60it/s]
[Chunk 8] Emotion: 100%|██████████| 209/209 [00:26<00:00,  8.02it/s]
[Chunk 8] Bias ZS: 100%|██████████| 834/834 [06:13<00:00,  2.23it/s]
[Chunk 8] Cred: 100%|██████████| 209/209 [00:43<00:00,  4.77it/s]

Saved: /content/drive/MyDrive/Final
Project/run_outputs/scores_part_08.parquet
Processing chunk 9 (5000 rows) ...

[Chunk 9] Sentiment: 100%|██████████| 157/157 [00:43<00:00,  3.58it/s]
[Chunk 9] Emotion: 100%|██████████| 209/209 [00:26<00:00,  7.95it/s]
[Chunk 9] Bias ZS: 100%|██████████| 834/834 [06:15<00:00,  2.22it/s]
[Chunk 9] Cred: 100%|██████████| 209/209 [00:43<00:00,  4.77it/s]
```

```
Saved: /content/drive/MyDrive/Final
Project/run_outputs/scores_part_09.parquet
Processing chunk 10 (1282 rows) ...

[Chunk 10] Sentiment: 100%|██████████| 41/41 [00:11<00:00,  3.68it/s]
[Chunk 10] Emotion: 100%|██████████| 54/54 [00:06<00:00,  8.79it/s]
[Chunk 10] Bias ZS: 100%|██████████| 214/214 [01:35<00:00,  2.25it/s]
[Chunk 10] Cred: 100%|██████████| 54/54 [00:10<00:00,  4.99it/s]

Saved: /content/drive/MyDrive/Final
Project/run_outputs/scores_part_10.parquet
```

Combine Parts, Compute Unified Score, Save Full Results

```
# Combine
parts = sorted([p for p in os.listdir(RUN_DIR) if
p.startswith("scores_part_") and p.endswith(".parquet")])
assert parts, "No parts found. Run Cell 12 first."
full = pd.concat([pd.read_parquet(os.path.join(RUN_DIR, p)) for p in
parts], axis=0, ignore_index=True)
print("Combined shape:", full.shape)

# Unified score (using CONFIG weights)
req = ["sent_norm","emot_norm","bias_norm","cred_norm"]
if not all(c in full.columns for c in req):
    missing = [c for c in req if c not in full.columns]
    raise ValueError(f"Missing columns: {missing}")

full["unified_score"] = compute_unified(full, CONFIG["weights"])

# Save master files
FULL_PARQ = os.path.join(RUN_DIR, "full_scores.parquet")
FULL_CSV  = os.path.join(RUN_DIR, "full_scores.csv")
full.to_parquet(FULL_PARQ, index=False)
full.to_csv(FULL_CSV, index=False)
print("Saved:\n", FULL_PARQ, "\n", FULL_CSV)

display(full.head(3))
```

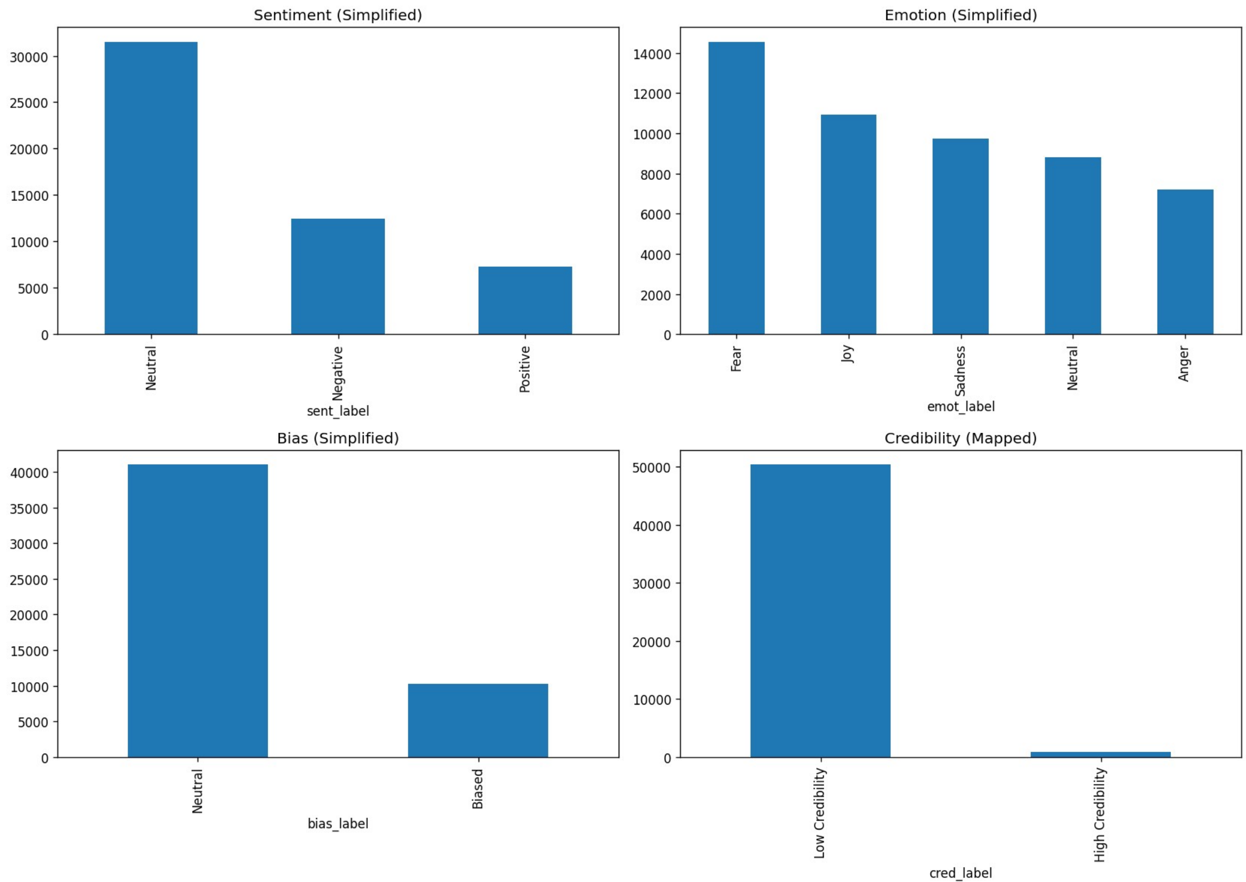
```
Combined shape: (51282, 24)
Saved:
 /content/drive/MyDrive/Final Project/run_outputs/full_scores.parquet
 /content/drive/MyDrive/Final Project/run_outputs/full_scores.csv
```

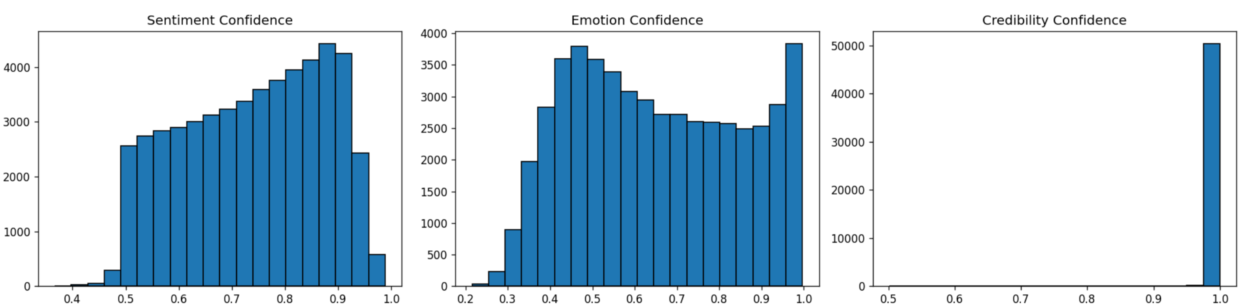
{"type":"dataframe"}

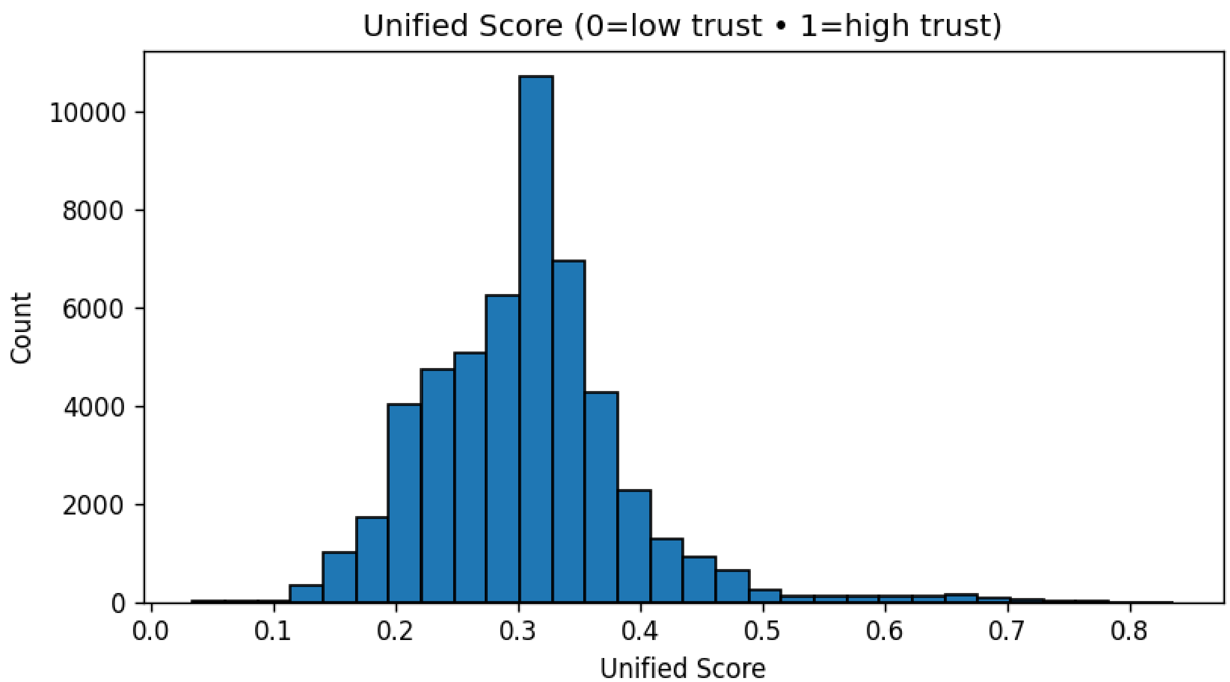
Core Visuals (and save PNGs)

```
# Label distributions
fig, axes = plt.subplots(2, 2, figsize=(14,10))
```

```
full["sent_label"].value_counts().plot(kind="bar", ax=axes[0,0],
title="Sentiment (Simplified)")
full["emot_label"].value_counts().plot(kind="bar", ax=axes[0,1],
title="Emotion (Simplified)")
full["bias_label"].value_counts().plot(kind="bar", ax=axes[1,0],
title="Bias (Simplified)")
full["cred_label"].value_counts().plot(kind="bar", ax=axes[1,1],
title="Credibility (Mapped)")
plt.tight_layout(); plt.savefig(os.path.join(EXPORTS,
"label_distributions.png")); plt.show()

# Confidence histograms (selected)
fig, axes = plt.subplots(1, 3, figsize=(16,4))
axes[0].hist(full["sent_conf"], bins=20, edgecolor="black");
axes[0].set_title("Sentiment Confidence")
axes[1].hist(full["emot_conf"], bins=20, edgecolor="black");
axes[1].set_title("Emotion Confidence")
axes[2].hist(full["cred_conf"], bins=20, edgecolor="black");
axes[2].set_title("Credibility Confidence")
plt.tight_layout(); plt.savefig(os.path.join(EXPORTS,
"confidence_histograms.png")); plt.show()

# Unified score distribution
plt.figure(figsize=(7,4))
plt.hist(full["unified_score"], bins=30, edgecolor="black")
plt.title("Unified Score (0=low trust • 1=high trust)");
plt.xlabel("Unified Score"); plt.ylabel("Count")
plt.tight_layout(); plt.savefig(os.path.join(EXPORTS,
"unified_score_hist.png")); plt.show()
```


Sentiment (Simplified)
30000
25000
20000
15000
10000
5000
0
Neutral
Negative
Positive
sent_label
Emotion (Simplified)
14000
12000
10000
8000
6000
4000
2000
0
Fear
Joy
Sadness
Neutral
Anger
emot_label
Bias (Simplified)
40000
35000
30000
25000
20000
15000
10000
5000
0
Neutral
Biased
bias_label
Credibility (Mapped)
50000
40000
30000
20000
10000
0
Low Credibility
High Credibility
cred_label


Sentiment Confidence
4000
3000
2000
1000
0
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Emotion Confidence
4000
3500
3000
2500
2000
1500
1000
500
0
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Credibility Confidence
50000
40000
30000
20000
10000
0
0.5
0.6
0.7
0.8
0.9
1.0



Category/Subcategory Summaries (CSV exports)

```
# Category means
cat_summary = (full.groupby("category")
[["sent_norm","emot_norm","bias_norm","cred_norm","unified_score"]]
               .mean().sort_values("unified_score", ascending=False))
cat_csv = os.path.join(EXPORTS, "category_summary.csv")
cat_summary.to_csv(cat_csv)
print("Saved:", cat_csv)
display(cat_summary.head(10))

# Top 15 subcategories by count
top_subs = full["subcategory"].value_counts().head(15).index
sub_summary = (full[full["subcategory"].isin(top_subs)]
               .groupby("subcategory")
[["sent_norm","emot_norm","bias_norm","cred_norm","unified_score"]]
               .mean().sort_values("unified_score", ascending=False))
sub_csv = os.path.join(EXPORTS, "subcategory_summary_top15.csv")
sub_summary.to_csv(sub_csv)
print("Saved:", sub_csv)
display(sub_summary)
```

```
Saved: /content/drive/MyDrive/Final
Project/exports/category_summary.csv
```

{"summary":"{\n \"name\": \"display(sub_summary)\",\n \"rows\": 10,\
n \"fields\": [\n   {\n   \"column\": \"category\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n   \"samples\": [\n

\"travel\",\n   \"kids\",\n   \"lifestyle\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"sent_norm\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.037602159445129534,\n   \"min\": 0.47527795328869865,\n   \"max\": 0.6109181997216201,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   0.532539980297393,\n   0.5546756316633785,\n   0.5602232778971596\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"emot_norm\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.048807149182894444,\n   \"min\": 0.43904266383881535,\n   \"max\": 0.590726912021637,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   0.4779932575580946,\n   0.5031456375823301,\n   0.4913052821288965\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"bias_norm\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.014949289675937 13,\n   \"min\": 0.4221169054508209,\n   \"max\": 0.47405020597547587,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   0.4469329144980045,\n   0.4681716070455663,\n   0.45736149174641005\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"cred_norm\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.027424815411074912,\n   \"min\": 0.0013173871389645735,\n   \"max\": 0.08945227315811854,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   0.007237294633337792,\n   0.0015125555150649127,\n   0.0026914651185189972\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"unified_score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.010334244502864725,\n   \"min\": 0.31259048121282174,\n   \"max\": 0.3515286069950726,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   0.316372929317 2669,\n   0.32913655004080605,\n   0.3255880977332954\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

Saved: /content/drive/MyDrive/Final Project/exports/subcategory_summary_top15.csv

{"summary":"{\n \"name\": \"sub_summary\",\n \"rows\": 15,\n \"fields\": [\n   {\n   \"column\": \"subcategory\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 15,\n   \"samples\": [\n   \"newspolitics\",\n   \"travelnews\",\n   \"newstrends\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"sent_norm\",\n   \"properties\": {\n   \"dtype\":

```
\"number\",\n        \"std\": 0.04543339768974934,\n        \"min\":
0.3907646686185238,\n        \"max\": 0.5657863375179621,\n
\"num_unique_values\": 15,\n        \"samples\": [\n
0.4670451694950514,\n          0.47932574708022985,\n
0.5657863375179621\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"emot_norm\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.060633983955997374,\n        \"min\":
0.319315513796484,\n        \"max\": 0.5590447192467876,\n
\"num_unique_values\": 15,\n        \"samples\": [\n
0.4052443723532074,\n          0.4422189850882604,\n
0.5590447192467876\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"bias_norm\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.009177150731284639,\n        \"min\":
0.4330277610652087,\n        \"max\": 0.47021289594303756,\n
\"num_unique_values\": 15,\n        \"samples\": [\n
0.46274439584508653,\n          0.4476248132614233,\n
0.45180381155338417\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cred_norm\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.04178343510834158,\n        \"min\":
0.002077940484540274,\n        \"max\": 0.16832207424003023,\n
\"num_unique_values\": 15,\n        \"samples\": [\n
0.03814810813949704,\n          0.01140716658727557,\n
0.006779959615395993\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"unified_score\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.01895327597763838,\n        \"min\":
0.258410742949764,\n        \"max\": 0.3402901501066846,\n
\"num_unique_values\": 15,\n        \"samples\": [\n
0.30349584517974737,\n          0.30020765805460037,\n
0.3402901501066846\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"sub_summary"}
```


Top/Bottom Articles + Radar Plots (saved)

```
cols_show = ["news_id","category","subcategory","title","abstract",
             "sent_label","emot_label","bias_label","cred_label",

"sent_norm","emot_norm","bias_norm","cred_norm","unified_score"]

top10 = full.sort_values("unified_score", ascending=False).head(10)
[cols_show]
bot10 = full.sort_values("unified_score", ascending=True).head(10)
[cols_show]

top10_path = os.path.join(EXPORTS, "top10_unified.csv")
```

```
bot10_path = os.path.join(EXPORTS, "bottom10_unified.csv")
top10.to_csv(top10_path, index=False); bot10.to_csv(bot10_path,
index=False)
print("Saved:\n", top10_path, "\n", bot10_path)
display(top10); display(bot10)

# Radar plot helper
def radar_plot(row, file_path):
    labels = ["Sentiment","Emotion","Neutrality","Credibility"]
    values = [row["sent_norm"], row["emot_norm"], row["bias_norm"],
row["cred_norm"]]
    values = values + values[:1]
    angles = np.linspace(0, 2*np.pi, len(labels),
endpoint=False).tolist()
    angles += angles[:1]

    fig = plt.figure(figsize=(5,5))
    ax = plt.subplot(111, polar=True)
    ax.plot(angles, values, linewidth=2)
    ax.fill(angles, values, alpha=0.25)
    ax.set_thetagrids(np.degrees(angles[:-1]), labels)
    ax.set_ylim(0, 1)
    ax.set_title(f"{row.get('news_id','')} •
Unified={row['unified_score']:.2f}")
    plt.tight_layout(); plt.savefig(file_path); plt.close(fig)

# Save radar for top1 and bottom1
radar_plot(top10.iloc[0], os.path.join(EXPORTS, "radar_top1.png"))
radar_plot(bot10.iloc[0], os.path.join(EXPORTS, "radar_bottom1.png"))
print("Saved radar plots.")
```

```
Saved:
 /content/drive/MyDrive/Final Project/exports/top10_unified.csv
 /content/drive/MyDrive/Final Project/exports/bottom10_unified.csv
```

{"summary":"{\n \"name\": \"top10\",\n \"rows\": 10,\n \"fields\":
[\n   {\n   \"column\": \"news_id\",\n   \"properties\": {\n
\"dtype\": \"string\",\n   \"num_unique_values\": 10,\n
\"samples\": [\n   \"N59163\",\n   \"N26376\",\n
\"N62124\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"category\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   \"sports\",\n   \"finance\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"subcategory\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 4,\n   \"samples\": [\n
\"finance-companies\",\n   \"finance-top-stocks\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\

n   },\n   {\n   \"column\": \"title\",\n   \"properties\": {\
n   \"dtype\": \"string\",\n   \"num_unique_values\": 10,\n
\"samples\": [\n   \"Banks reap $1 billion from US mortgage
bond trading boom\",\n   \"Stocks close higher on optimism over
China trade deal\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"abstract\",\n   \"properties\": {\n   \"dtype\":
\"string\",\n   \"num_unique_values\": 10,\n   \"samples\":
[\n   \"Global banks earned $1 billion from trading government-
backed U.S. mortgage securities in the first half of 2019, data shows,
a fivefold increase over last year for what industry sources say is
the fastest-growing revenue source in investment banking.\",\n
\"Stocks ended at record highs Thursday after the world's two largest
economies reportedly agreed to remove existing trade tariffs, sparking
a huge rotation into equities and out of bonds.\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"sent_label\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n   \"samples\": [\n
\"Neutral\",\n   \"Positive\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"emot_label\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 1,\n   \"samples\": [\n   \"Joy\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"bias_label\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n   \"samples\": [\n
\"Biased\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"cred_label\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 1,\n   \"samples\":
[\n   \"High Credibility\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"sent_norm\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0.1410927875127193,\n   \"min\": 0.5,\n   \"max\":
0.9615849852561951,\n   \"num_unique_values\": 10,\n
\"samples\": [\n   0.7645527720451355\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"emot_norm\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0.11457000922332933,\n   \"min\": 0.6375457048416138,\n
\"max\": 0.9876819252967834,\n   \"num_unique_values\": 10,\n
\"samples\": [\n   0.8598490357398987\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"bias_norm\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0.09731568383921953,\n   \"min\": 0.39949920773506165,\n

```
\"max\": 0.6962983012199402,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          0.39949920773506165\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"cred_norm\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.006885270787107911,\n        \"min\": 0.9842750430107117,\n
\"max\": 0.9998519420623779,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          0.9997043013572693\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"unified_score\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.023451693198883013,\n        \"min\": 0.7737273454666138,\n
\"max\": 0.8361915156245232,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          0.7746516689658165\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"top10"}

{"summary":"{\n  \"name\": \"bot10\",\n  \"rows\": 10,\n  \"fields\":
[\n    {\n      \"column\": \"news_id\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 10,\n
\"samples\": [\n          \"N21433\",\n          \"N3428\",\n
\"N4612\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"category\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          \"lifestyle\",\n          \"finance\",\n
\"news\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"subcategory\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          \"newsopinion\",\n          \"newsus\",\n
\"newsworld\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"title\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n        \"samples\": [\n          \"A
Halloween message from Gov. Jay Inslee: I'm 'pretty scared'\",\n
\"'People Are Going To Get Injured, Or Worse': Animal Rights Activists
Claim Responsibility For Wis. Deer Stands Vandalism\",\n
\"MLB needs to step up on cheating allegations against Astros: 'It's a
serious matter'\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"abstract\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          \"Gov. Jay Inslee is trolling for campaign donations by
Halloween night, and has sent out an email blast headlined: \\\"I'm
pretty scared right now.\\\" The dropout presidential candidate is
raising money in his quest for a third term as Governor, and using a
tactic that is the money's milk of political fundraising   fear. He's
conjuring up the prospect of defeat, although Democrats have won every
gubernatorial race in Washington since 1980....\",\n
```

```
\"Authorities in western Wisconsin are investigating after a number of
deer stands were vandalized ahead of hunting season.\",\n
\"Veteran righty Mike Fiers said that the Astros were stealing signs
using electronic means during the 2017 season, another blemish for the
organization.\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"sent_label\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"Negative\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"emot_label\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"Fear\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"bias_label\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n
\"Biased\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cred_label\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"Low Credibility\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"sent_norm\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.043798972085419455,\n        \"min\": 0.05841797590255737,\n
\"max\": 0.22689735889434814,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          0.18092143535614014\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"emot_norm\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.026167982480413517,\n        \"min\": 0.011786103248596191,\n
\"max\": 0.09069913625717163,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          0.031071025412529707\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"bias_norm\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.03614469254828408,\n        \"min\": 0.019779562950134277,\n
\"max\": 0.13914334774017334,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          0.10224413871765137\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"cred_norm\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.006382910745073452,\n        \"min\": 0.0008757710456848145,\n
\"max\": 0.021527409553527832,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          0.0017758011817932129\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"unified_score\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
```

```
0.011015136914955271,\n        \"min\": 0.034151193546131255,\n
\"max\": 0.06993071793112904,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          0.06858105724677445\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"bot10"}
```


```
Saved radar plots.
```

Exports

```python
# Compact deliverable with all signals
deliver_cols = [
    "news_id","category","subcategory","title","abstract",
    "sent_label","sent_conf","sent_norm",
    "emot_label","emot_conf","emot_norm",
    "bias_label","bias_conf","bias_norm",
    "cred_label","cred_conf","cred_norm",
    "unified_score"
]
deliverable = full[deliver_cols]
DELIVER_CSV = os.path.join(EXPORTS, "deliverable_scores.csv")
deliverable.to_csv(DELIVER_CSV, index=False)
print("Saved:", DELIVER_CSV)
display(deliverable.head(5))

# Small samples for quick checking
data_sample_path = os.path.join(EXPORTS, "sample_processed_text.csv")
pd.DataFrame({
    "news_id": full["news_id"].head(20),
    "title": full["title"].head(20),
    "abstract": full["abstract"].head(20),
    "processed_text": full["processed_text"].head(20)
}).to_csv(data_sample_path, index=False)

pd.DataFrame({
    "news_id": full["news_id"].head(50),
    "processed_text": full["processed_text"].head(50),
    "sent_label": full["sent_label"].head(50),
    "sent_conf": full["sent_conf"].head(50),
    "sent_norm": full["sent_norm"].head(50)
}).to_csv(os.path.join(EXPORTS, "sentiment_results.csv"), index=False)

pd.DataFrame({
    "news_id": full["news_id"].head(50),
    "processed_text": full["processed_text"].head(50),
    "emot_label": full["emot_label"].head(50),
    "emot_conf": full["emot_conf"].head(50),
    "emot_norm": full["emot_norm"].head(50)
}).to_csv(os.path.join(EXPORTS, "emotion_results.csv"), index=False)
```

```
pd.DataFrame({
    "news_id": full["news_id"].head(50),
    "processed_text": full["processed_text"].head(50),
    "bias_label": full["bias_label"].head(50),
    "bias_conf": full["bias_conf"].head(50),
    "bias_norm": full["bias_norm"].head(50)
}).to_csv(os.path.join(EXPORTS, "bias_results.csv"), index=False)

pd.DataFrame({
    "news_id": full["news_id"].head(50),
    "processed_text": full["processed_text"].head(50),
    "cred_label": full["cred_label"].head(50),
    "cred_conf": full["cred_conf"].head(50),
    "cred_norm": full["cred_norm"].head(50)
}).to_csv(os.path.join(EXPORTS, "credibility_results.csv"),
index=False)

pd.DataFrame({
    "news_id": full["news_id"].head(50),
    "sent_norm": full["sent_norm"].head(50),
    "emot_norm": full["emot_norm"].head(50),
    "bias_norm": full["bias_norm"].head(50),
    "cred_norm": full["cred_norm"].head(50),
    "unified_score": full["unified_score"].head(50)
}).to_csv(os.path.join(EXPORTS, "unified_score_results.csv"),
index=False)

print("Saved sample CSVs to:", EXPORTS)
```

```
Saved: /content/drive/MyDrive/Final
Project/exports/deliverable_scores.csv
```

{"summary":"{\n \"name\": \"print(\\\"Saved sample CSVs to:\\\",
EXPORTS)\",\n \"rows\": 5,\n \"fields\": [\n   {\n   \"column\":
\"news_id\",\n   \"properties\": {\n   \"dtype\": \"string\",\
n   \"num_unique_values\": 5,\n   \"samples\": [\n
\"N19639\",\n   \"N38324\",\n   \"N61837\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"category\",\n   \"properties\":
{\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3,\n
\"samples\": [\n   \"lifestyle\",\n   \"health\",\n
\"news\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"subcategory\",\n   \"properties\": {\n   \"dtype\":
\"string\",\n   \"num_unique_values\": 5,\n   \"samples\":
[\n   \"weightloss\",\n   \"medical\",\n
\"newsworld\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"title\",\n   \"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n   \"50

Worst Habits For Belly Fat\",\n   \"How to Get Rid of Skin Tags, According to a Dermatologist\",\n   \"The Cost of Trump's Aid Freeze in the Trenches of Ukraine's War\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"abstract\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 5,\n   \"samples\": [\n   \"These seemingly harmless habits are holding you back and keeping you from shedding that unwanted belly fat for good.\",\n   \"They seem harmless, but there's a very good reason you shouldn't ignore them. The post How to Get Rid of Skin Tags, According to a Dermatologist appeared first on Reader's Digest.\",\n   \"Lt. Ivan Molchanets peeked over a parapet of sand bags at the front line of the war in Ukraine. Next to him was an empty helmet propped up to trick snipers, already perforated with multiple holes.\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"sent_label\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"Negative\",\n   \"Neutral\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"sent_conf\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.11746145530671052,\n   \"min\": 0.6301542520523071,\n   \"max\": 0.9206874370574951,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   0.7141546607017517,\n   0.6372511386871338\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"sent_norm\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.17516876482451252,\n   \"min\": 0.07931256294250488,\n   \"max\": 0.5,\n   \"num_unique_values\": 4,\n   \"samples\": [\n   0.2858453392982483,\n   0.07931256294250488\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"emot_label\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 4,\n   \"samples\": [\n   \"Anger\",\n   \"Neutral\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"emot_conf\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.25805784780690344,\n   \"min\": 0.2694278284907341,\n   \"max\": 0.8972344174981117,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   0.8972344174981117,\n   0.7912735342979431\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"emot_norm\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.2363587134887013,\n   \"min\": 0.10276558250188828,\n   \"max\": 0.7305721715092659,\n   \"num_unique_values\": 5,\n   \"samples\": [\n

```
0.10276558250188828,\n          0.6\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"bias_label\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n
\"Neutral\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"bias_conf\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.048879485789632394,\n        \"min\":
0.370210736989975,\n        \"max\": 0.4857232868671417,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.4857232868671417\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"bias_norm\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.048879485789632394,\n        \"min\":
0.370210736989975,\n        \"max\": 0.4857232868671417,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.4857232868671417\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cred_label\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"Low Credibility\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"cred_conf\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.0006031852355311741,\n        \"min\": 0.9977549910545349,\n
\"max\": 0.9991850256919861,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.9990267753601074\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"cred_norm\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.0006031852355311742,\n        \"min\": 0.000814974308013916,\n
\"max\": 0.002245008945465088,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.0009732246398925781\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"unified_score\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.06283102547891092,\n        \"min\": 0.19949363470077514,\n
\"max\": 0.33304530948400496,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.19949363470077514\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


```
Saved sample CSVs to: /content/drive/MyDrive/Final Project/exports
```

Manual-Label Template

```
# Random 200 for manual labeling
sample_for_labels = full.sample(200, random_state=SEED)[
```

```
    ["news_id","category","subcategory","title","abstract"]
].copy()

sample_for_labels["man_sentiment"]   = ""   # Positive / Neutral /
Negative
sample_for_labels["man_emotion"]     = ""   # Joy / Sadness / Anger /
Fear / Neutral
sample_for_labels["man_bias"]        = ""   # Biased / Neutral
sample_for_labels["man_credibility"] = ""   # High Credibility / Low
Credibility

TEMPLATE_PATH = os.path.join(EXPORTS, "manual_label_template.csv")
sample_for_labels.to_csv(TEMPLATE_PATH, index=False)
print("Saved manual label template:", TEMPLATE_PATH)
display(sample_for_labels.head(3))
```

```
Saved manual label template: /content/drive/MyDrive/Final
Project/exports/manual_label_template.csv
```

{"repr_error":"0","type":"dataframe"}

```
import os, pandas as pd

BASE_DIR = "/content/drive/MyDrive/Final Project"
RUN_DIR  = os.path.join(BASE_DIR, "run_outputs")

print("Files in run_outputs:", sorted(os.listdir(RUN_DIR))[:10])

full_parq = os.path.join(RUN_DIR, "full_scores.parquet")
full_csv  = os.path.join(RUN_DIR, "full_scores.csv")
print("full_scores.parquet exists:", os.path.exists(full_parq))
print("full_scores.csv exists:", os.path.exists(full_csv))

# Load one of them to preview
# If you ever hit a pyarrow error, add: engine="fastparquet"
full = pd.read_parquet(full_parq)
full.head(3)
```

```
Files in run_outputs: ['full_scores.csv', 'full_scores.parquet',
'processed_text.parquet', 'scores_part_00.parquet',
'scores_part_01.parquet', 'scores_part_02.parquet',
'scores_part_03.parquet', 'scores_part_04.parquet',
'scores_part_05.parquet', 'scores_part_06.parquet']
full_scores.parquet exists: True
full_scores.csv exists: True
```

{"type":"dataframe","variable_name":"full"}

Create a stratified 100-row manual-label set

```
# === Make a stratified random sample of 100 rows for manual labeling
===
import os, math
import pandas as pd
import numpy as np

BASE_DIR = "/content/drive/MyDrive/Final Project"
EXPORTS  = os.path.join(BASE_DIR, "exports")

# Load the already-produced deliverable (has all text + model preds)
df = pd.read_csv(os.path.join(EXPORTS, "deliverable_scores.csv"))

SEED = 42
np.random.seed(SEED)

# target size
N_TARGET = 100

# Compute proportional allocation per category (at least 2 per
category)
cat_counts = df["category"].value_counts()
cat_props  = cat_counts / cat_counts.sum()
alloc = (cat_props * N_TARGET).round().astype(int).clip(lower=2)

# adjust total to exactly 100
diff = N_TARGET - alloc.sum()
if diff > 0:
    # add 1 to the largest categories until we hit 100
    for cat in cat_counts.index:
        if diff == 0: break
        alloc[cat] += 1
        diff -= 1
elif diff < 0:
    # remove 1 from the largest categories until we hit 100 (but keep
>=2)
    for cat in cat_counts.index:
        if diff == 0: break
        if alloc[cat] > 2:
            alloc[cat] -= 1
            diff += 1

# sample per category
parts = []
for cat, k in alloc.items():
    sub = df[df["category"] == cat]
    take = min(k, len(sub))
    parts.append(sub.sample(take, random_state=SEED))
```

```
man100 = pd.concat(parts, ignore_index=True)

# Keep only the columns needed for human judgment
keep_cols = ["news_id","category","subcategory","title","abstract"]
man100 = man100[keep_cols].drop_duplicates().reset_index(drop=True)

# Add empty manual label columns
man100["man_sentiment"]   = ""   # Positive / Neutral / Negative
man100["man_emotion"]     = ""   # Joy / Sadness / Anger / Fear /
Neutral
man100["man_bias"]        = ""   # Biased / Neutral
man100["man_credibility"] = ""   # High Credibility / Low Credibility

out_path = os.path.join(EXPORTS, "manual_label_set_100.csv")
man100.to_csv(out_path, index=False)
print("Saved stratified manual set to:", out_path)
man100.head(5)
```

```
Saved stratified manual set to: /content/drive/MyDrive/Final
Project/exports/manual_label_set_100.csv
```


{"summary":"{\n \"name\": \"man100\",\n \"rows\": 99,\n \"fields\":
[\n   {\n   \"column\": \"news_id\",\n   \"properties\": {\n
\"dtype\": \"string\",\n   \"num_unique_values\": 99,\n
\"samples\": [\n   \"N39529\",\n   \"N27212\",\n
\"N41051\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"category\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 17,\n
\"samples\": [\n   \"news\",\n   \"sports\",\n
\"travel\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"subcategory\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 45,\n
\"samples\": [\n   \"movies-celebrity\",\n
\"traveltripideas\",\n   \"travelarticle\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"title\",\n   \"properties\": {\
n   \"dtype\": \"string\",\n   \"num_unique_values\": 99,\n
\"samples\": [\n   \"Make Ryan Scott's easy one-pot beef
stew\",\n   \"Instant analysis of Seattle's 32-28 victory over
Cleveland\",\n   \"Take those pumpkin seeds, and make a
chocolaty treat\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"abstract\",\n   \"properties\": {\n   \"dtype\":
\"string\",\n   \"num_unique_values\": 89,\n   \"samples\":
[\n   \"Gameday is here and we've got everything you need to
know\",\n   \"According to the 2018 United Nations Human
Development Index   a composite measure of education, life expectancy,
and standard of living   the United States is the 13th most developed

```
country in the world, behind nations such as Norway, Switzerland, and
Australia.\",\n          \"A daily look at hockey news around the
world.\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"man_sentiment\",\n      \"properties\": {\n        \"dtype\":
\"object\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"man_emotion\",\n      \"properties\": {\n        \"dtype\":
\"object\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"man_bias\",\n      \"properties\": {\n        \"dtype\":
\"object\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"man_credibility\",\n      \"properties\": {\n        \"dtype\":
\"object\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"man100"}
```


Evaluation (manual label)

```
import pandas as pd
import numpy as np
import ast, json, re
from sklearn.metrics import classification_report, confusion_matrix,
cohen_kappa_score, f1_score, accuracy_score
from collections import Counter

# ========= CONFIG =========
MANUAL_PATH = "/content/drive/MyDrive/Final
Project/manual_label_set_100_labeled.csv"
SCORED_PATH = "/content/drive/MyDrive/Final
Project/run_outputs/full_scores.csv"
ID_COL = "news_id"

# ========= LOAD =========
man = pd.read_csv(MANUAL_PATH)
scored = pd.read_csv(SCORED_PATH)

# Sanity
needed_cols = {ID_COL, "man_sentiment", "man_emotion", "man_bias",
"man_credibility"}
missing = [c for c in needed_cols if c not in man.columns]
if missing:
    raise ValueError(f"Manual file missing columns: {missing}")
```

```
df = man.merge(scored, on=ID_COL, how="left")
print(f"Merged rows: {len(df)} (manual={len(man)},
scored={len(scored)})")

# ========= NORMALIZATION HELPERS =========
def norm_label(x):
    if pd.isna(x): return None
    s = str(x).strip().lower()
    s = re.sub(r"\s+", " ", s)
    # unify common variants
    mapping = {
        "pos": "positive", "positive": "positive",
        "neg": "negative", "negative": "negative",
        "neu": "neutral", "neutral": "neutral",
        "joy": "joy", "sadness": "sadness", "anger": "anger", "fear":
"fear",
        "surprise": "surprise", "neutral emotion": "neutral",
        "biased": "biased", "bias": "biased", "neutral bias":
"neutral",
        "high credibility": "high credibility", "low credibility":
"low credibility",
        "high": "high credibility", "low": "low credibility",
    }
    return mapping.get(s, s)

def find_col(candidates, cols):
    for c in candidates:
        if c in cols:
            return c
    return None

# Try to find prediction columns in scored file
cols = set(df.columns)

# Sentiment prediction column options
PRED_SENT_COL = find_col(

["sentiment_label","sent_label","pred_sentiment","sentiment_class","se
ntiment","pred_sent"],
    cols
)

# Emotion prediction: either a single label or a top-list like
"[(label, prob), ...]"
PRED_EMOT_COL = find_col(

["emot_label","emotion_label","emotion","pred_emotion","emotions_top",
"emotions"],
    cols
)
```

```
# Bias prediction (categorical)
PRED_BIAS_COL = find_col(
    ["bias_label","pred_bias","bias","bias_class"],
    cols
)

# Credibility prediction (categorical)
PRED_CRED_COL = find_col(

["cred_label","pred_credibility","credibility_label","credibility_clas
s","credibility"],
    cols
)

print("Detected prediction columns:")
print("  Sentiment  ->", PRED_SENT_COL)
print("  Emotion    ->", PRED_EMOT_COL)
print("  Bias       ->", PRED_BIAS_COL)
print("  Credibility->", PRED_CRED_COL)

# If emotions_top is a JSON-ish list, extract top-1 label
def get_emotion_top1(val):
    if pd.isna(val): return None
    s = str(val).strip()
    # Try parsing Python-list-like or JSON
    try:
        obj = ast.literal_eval(s)
    except Exception:
        try:
            obj = json.loads(s)
        except Exception:
            return norm_label(s)  # already a single label?
    # obj may look like [("joy", 0.87), ("admiration", 0.4), ...] or
[{"label":"joy","score":0.87}, ...]
    if isinstance(obj, list) and len(obj) > 0:
        first = obj[0]
        if isinstance(first, (list, tuple)) and len(first) >= 1:
            return norm_label(first[0])
        if isinstance(first, dict):
            # look for label key
            if "label" in first:
                return norm_label(first["label"])
            # could be {"joy":0.87}
            key = list(first.keys())[0]
            return norm_label(key)
    return None

def safe_series_top1(series):
    return series.apply(get_emotion_top1)
```

```python
# Prepare ground-truth columns (normalized)
df["gt_sentiment"]   = df["man_sentiment"].apply(norm_label)
df["gt_emotion"]     = df["man_emotion"].apply(norm_label)
df["gt_bias"]        = df["man_bias"].apply(norm_label)
df["gt_credibility"] = df["man_credibility"].apply(norm_label)

# Prepare prediction columns (normalized)
if PRED_SENT_COL:
    df["pr_sentiment"] = df[PRED_SENT_COL].apply(norm_label)
if PRED_EMOT_COL:
    if "top" in PRED_EMOT_COL or
df[PRED_EMOT_COL].astype(str).str.startswith("[").any():
        df["pr_emotion"] = safe_series_top1(df[PRED_EMOT_COL])
    else:
        df["pr_emotion"] = df[PRED_EMOT_COL].apply(norm_label)
if PRED_BIAS_COL:
    df["pr_bias"] = df[PRED_BIAS_COL].apply(norm_label)
if PRED_CRED_COL:
    df["pr_credibility"] = df[PRED_CRED_COL].apply(norm_label)

# Small peek
df[[ID_COL,"gt_sentiment","pr_sentiment","gt_emotion","pr_emotion","gt
_bias","pr_bias","gt_credibility","pr_credibility"]].head(8)
```

```
Merged rows: 99 (manual=99, scored=51282)
Detected prediction columns:
  Sentiment  -> sent_label
  Emotion    -> emot_label
  Bias       -> bias_label
  Credibility-> cred_label
```


```
{"summary":"{\n  \"name\":
\"df[[ID_COL,\\\"gt_sentiment\\\",\\\"pr_sentiment\\\",\\\"gt_emotion\
\\\",\\\"pr_emotion\\\",\\\"gt_bias\\\",\\\"pr_bias\\\",\\\"gt_credibil
ity\\\",\\\"pr_credibility\\\"]]\",\n  \"rows\": 8,\n  \"fields\": [\n
{\n      \"column\": \"news_id\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 8,\n
\"samples\": [\n          \"N27874\",\n          \"N50720\",\n
\"N27324\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"gt_sentiment\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"neutral\",\n          \"negative\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"pr_sentiment\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n
\"negative\",\n          \"neutral\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

```
n    },\n    {\n      \"column\": \"gt_emotion\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n
\"neutral\",\n          \"fear\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"pr_emotion\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n
\"fear\",\n          \"anger\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"gt_bias\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"biased\",\n          \"neutral\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"pr_bias\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
2,\n        \"samples\": [\n          \"biased\",\n
\"neutral\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"gt_credibility\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"low credibility\",\n          \"high credibility\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"pr_credibility\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n          \"low
credibility\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

from pprint import pprint

def eval_task(gt_col, pr_col, task_name, labels_order=None):
    print("\n" + "="*70)
    print(f"{task_name.upper()} — {gt_col} vs {pr_col}")
    sub = df[[gt_col, pr_col]].dropna()
    y_true = sub[gt_col].tolist()
    y_pred = sub[pr_col].tolist()

    if not y_true or not y_pred:
        print("No data to evaluate for this task.")
        return

    # If label order provided, ensure report follows it
    unique_labels = sorted(set(y_true) | set(y_pred)) if labels_order
is None else labels_order

    print("\nCounts:")
    print("  Ground truth:", Counter(y_true))
    print("  Predictions :", Counter(y_pred))
```

```
    print("\nClassification report (macro):")
    print(classification_report(y_true, y_pred, labels=unique_labels,
zero_division=0, digits=3))

    # Cohen's kappa (treat as nominal)
    try:
        kappa = cohen_kappa_score(y_true, y_pred)
        print(f"Cohen's κ: {kappa:.3f}")
    except Exception as e:
        print("Kappa error:", e)

    # Accuracy + macro F1
    acc = accuracy_score(y_true, y_pred)
    f1m = f1_score(y_true, y_pred, average="macro", zero_division=0)
    print(f"Accuracy: {acc:.3f} | Macro-F1: {f1m:.3f}")

    # Confusion matrix
    cm = confusion_matrix(y_true, y_pred, labels=unique_labels)
    print("\nLabels order:", unique_labels)
    print("Confusion matrix (rows=GT, cols=Pred):")
    print(cm)

# Run evaluations (only if those prediction cols exist)
if "pr_sentiment" in df.columns:
    eval_task("gt_sentiment", "pr_sentiment", "Sentiment",
labels_order=["negative","neutral","positive"])

if "pr_emotion" in df.columns:
    # Use a compact set; your GT set is typically one of these
    common_emotions =
["anger","fear","joy","sadness","surprise","neutral"]
    eval_task("gt_emotion", "pr_emotion", "Emotion",
labels_order=common_emotions)

if "pr_bias" in df.columns:
    eval_task("gt_bias", "pr_bias", "Bias",
labels_order=["neutral","biased"])

if "pr_credibility" in df.columns:
    eval_task("gt_credibility", "pr_credibility", "Credibility",
labels_order=["low credibility","high credibility"])
```

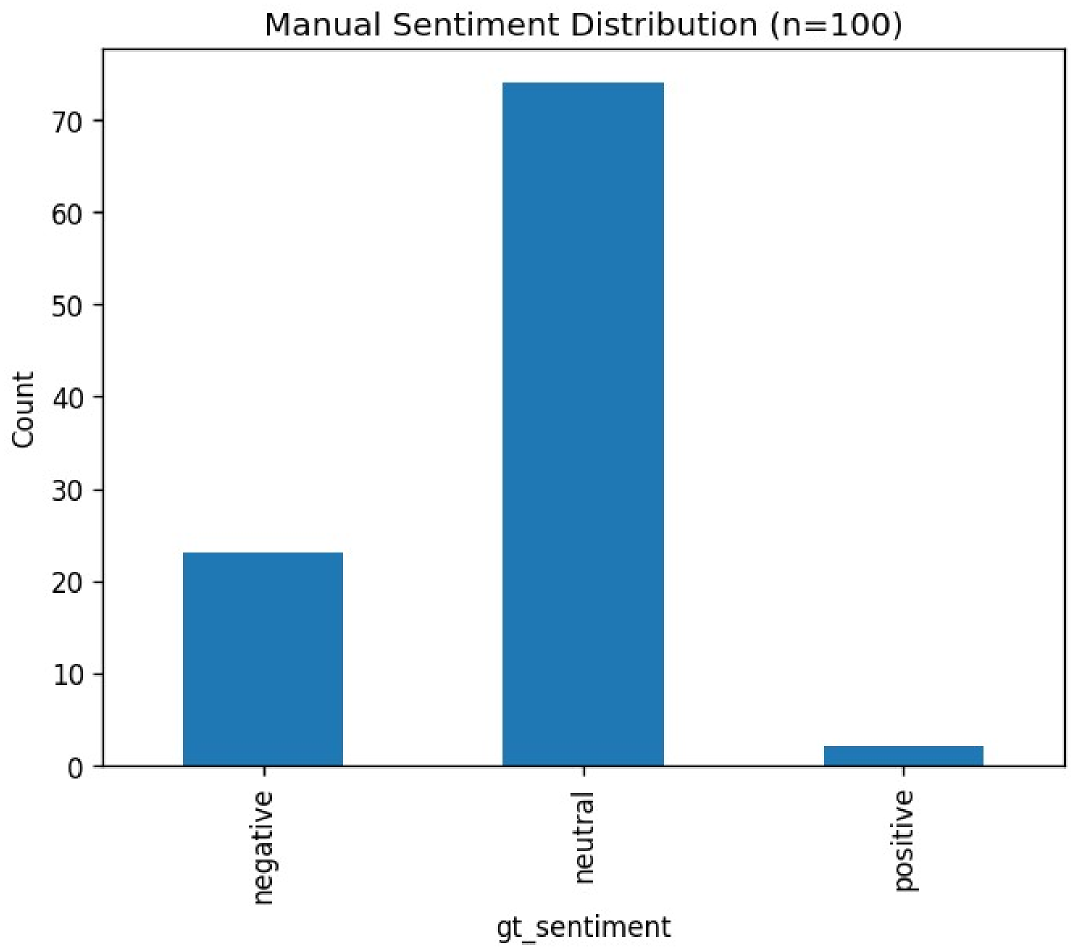
```
======================================================================
SENTIMENT — gt_sentiment vs pr_sentiment

Counts:
  Ground truth: Counter({'neutral': 74, 'negative': 23, 'positive':
2})
  Predictions : Counter({'neutral': 58, 'negative': 28, 'positive':
```

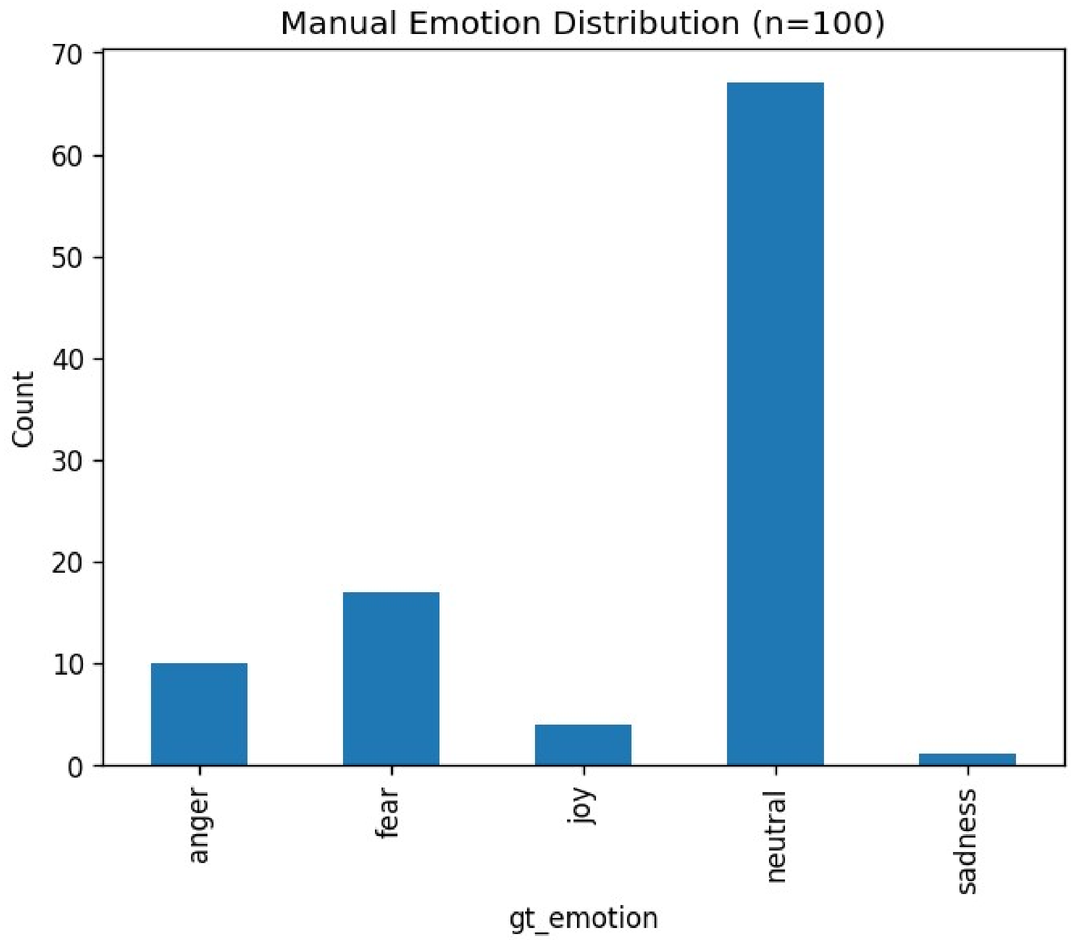
```
13})

Classification report (macro):
              precision    recall  f1-score   support

    negative      0.429     0.522     0.471        23
     neutral      0.793     0.622     0.697        74
    positive      0.077     0.500     0.133         2

    accuracy                          0.596        99
   macro avg      0.433     0.548     0.434        99
weighted avg      0.694     0.596     0.633        99

Cohen's κ: 0.182
Accuracy: 0.596 | Macro-F1: 0.434

Labels order: ['negative', 'neutral', 'positive']
Confusion matrix (rows=GT, cols=Pred):
[[12 11  0]
 [16 46 12]
 [ 0  1  1]]

======================================================================
EMOTION — gt_emotion vs pr_emotion

Counts:
  Ground truth: Counter({'neutral': 66, 'fear': 18, 'anger': 10,
'joy': 4, 'sadness': 1})
  Predictions : Counter({'fear': 22, 'joy': 21, 'neutral': 21,
'sadness': 18, 'anger': 17})

Classification report (macro):
              precision    recall  f1-score   support

       anger      0.235     0.400     0.296        10
        fear      0.273     0.333     0.300        18
         joy      0.143     0.750     0.240         4
     sadness      0.000     0.000     0.000         1
    surprise      0.000     0.000     0.000         0
     neutral      0.905     0.288     0.437        66

    accuracy                          0.323        99
   macro avg      0.259     0.295     0.212        99
weighted avg      0.682     0.323     0.385        99

Cohen's κ: 0.144
Accuracy: 0.323 | Macro-F1: 0.255

Labels order: ['anger', 'fear', 'joy', 'sadness', 'surprise',
'neutral']
```

```
Confusion matrix (rows=GT, cols=Pred):
[[ 4  2  2  2  0  0]
 [ 9  6  0  2  0  1]
 [ 0  0  3  0  0  1]
 [ 0  1  0  0  0  0]
 [ 0  0  0  0  0  0]
 [ 4 13 16 14  0 19]]

======================================================================
BIAS — gt_bias vs pr_bias

Counts:
  Ground truth: Counter({'neutral': 85, 'biased': 14})
  Predictions : Counter({'neutral': 75, 'biased': 24})

Classification report (macro):
              precision    recall  f1-score   support

     neutral      0.893     0.788     0.838        85
      biased      0.250     0.429     0.316        14

    accuracy                          0.737        99
   macro avg      0.572     0.608     0.577        99
weighted avg      0.802     0.737     0.764        99

Cohen's κ: 0.167
Accuracy: 0.737 | Macro-F1: 0.577

Labels order: ['neutral', 'biased']
Confusion matrix (rows=GT, cols=Pred):
[[67 18]
 [ 8  6]]

======================================================================
CREDIBILITY — gt_credibility vs pr_credibility

Counts:
  Ground truth: Counter({'high credibility': 96, 'low credibility':
3})
  Predictions : Counter({'low credibility': 95, 'high credibility':
4})

Classification report (macro):
                  precision    recall  f1-score   support

 low credibility      0.032     1.000     0.061         3
high credibility      1.000     0.042     0.080        96

        accuracy                          0.071        99
       macro avg      0.516     0.521     0.071        99
```

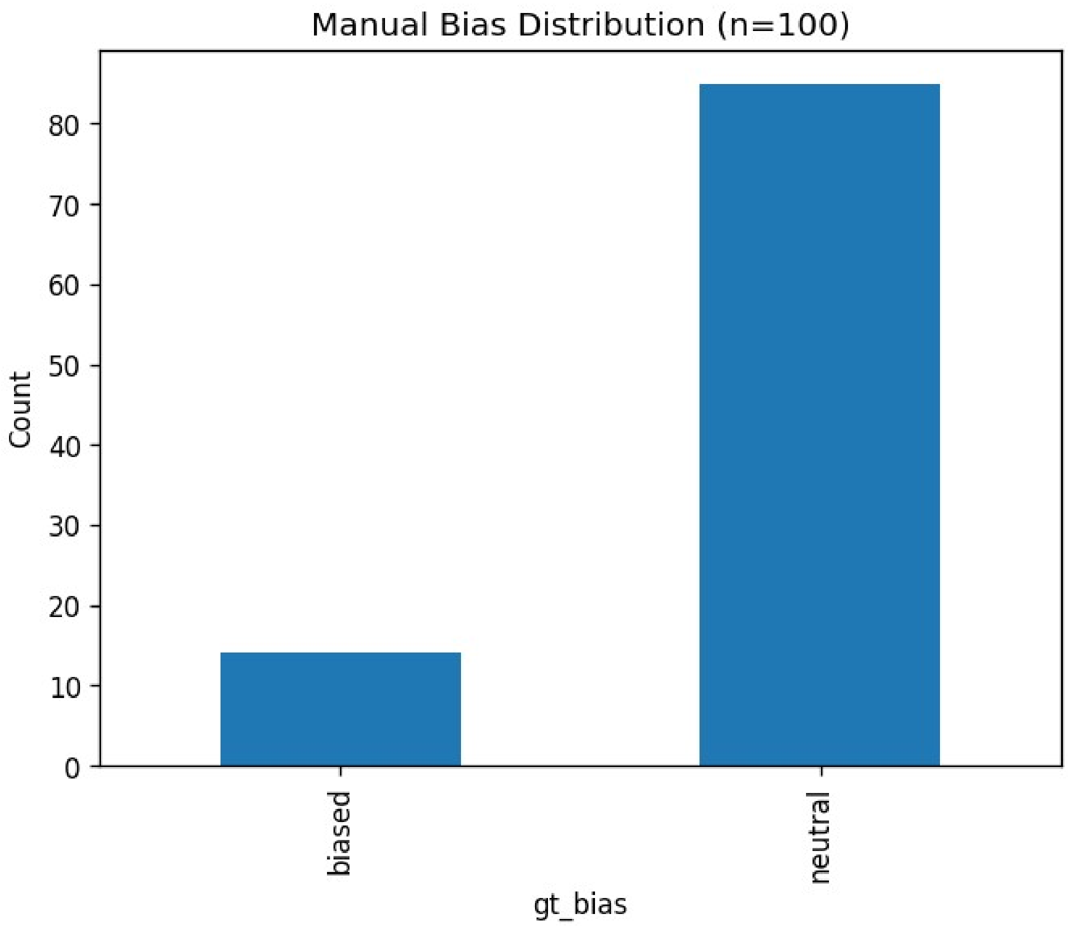
```
    weighted avg      0.971     0.071     0.079        99

Cohen's κ: 0.003
Accuracy: 0.071 | Macro-F1: 0.071

Labels order: ['low credibility', 'high credibility']
Confusion matrix (rows=GT, cols=Pred):
[[ 3  0]
 [92  4]]
```

```python
import matplotlib.pyplot as plt

def plot_dist(col, title):
    vc = df[col].value_counts().sort_index()
    vc.plot(kind="bar")
    plt.title(title)
    plt.xlabel(col)
    plt.ylabel("Count")
    plt.show()

# Distributions of your manual labels
plot_dist("gt_sentiment",   "Manual Sentiment Distribution (n=100)")
plot_dist("gt_emotion",     "Manual Emotion Distribution (n=100)")
plot_dist("gt_bias",        "Manual Bias Distribution (n=100)")
plot_dist("gt_credibility", "Manual Credibility Distribution (n=100)")
```

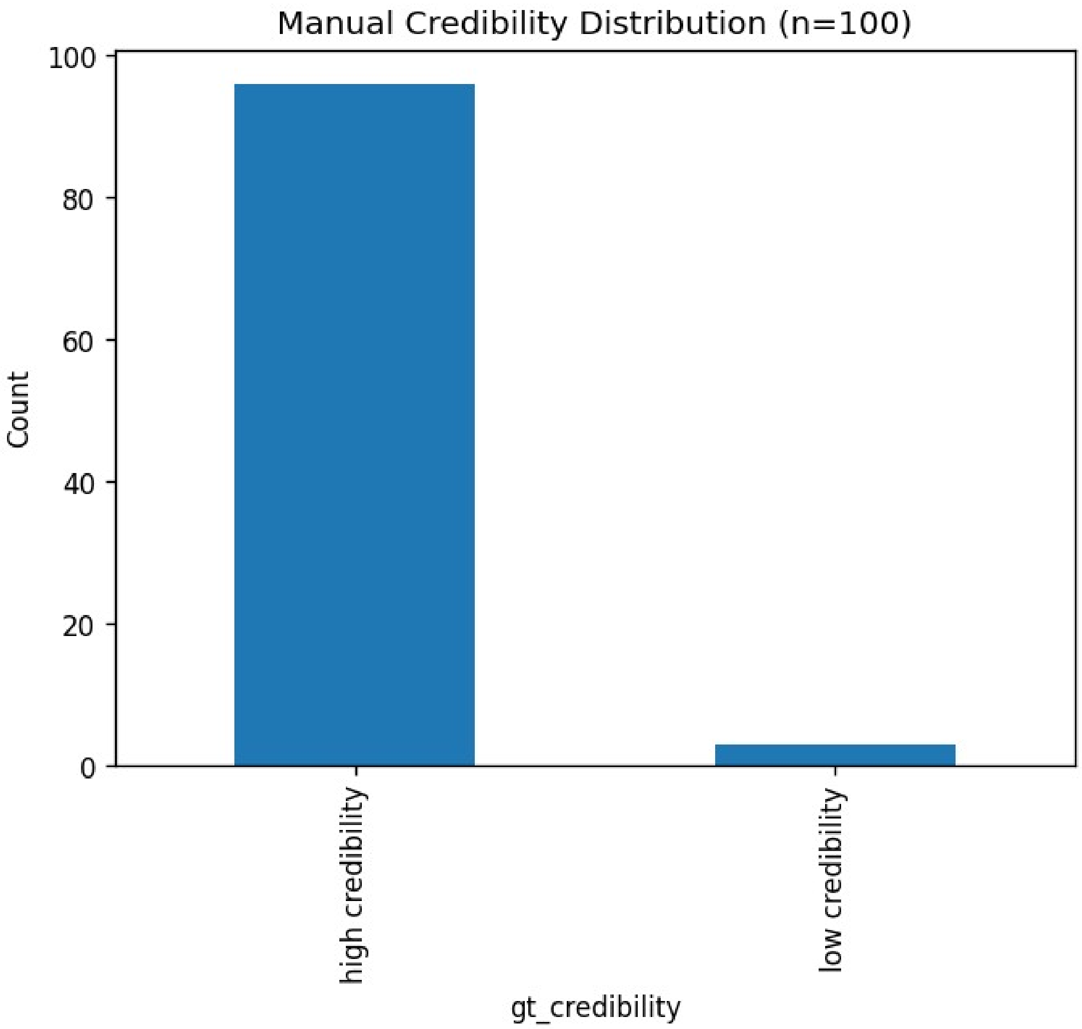

Manual Sentiment Distribution (n=100)
Count
0
10
20
30
40
50
60
70
negative
neutral
positive
gt_sentiment



Manual Emotion Distribution (n=100)
Count
0
10
20
30
40
50
60
70
anger
fear
joy
neutral
sadness
gt_emotion



Manual Bias Distribution (n=100)
Count
0
10
20
30
40
50
60
70
80
biased
neutral
gt_bias



Heuristic labeling (configure paths & thresholds once)

```
import pandas as pd
import numpy as np

# INPUT
FULL_SCORES = "/content/drive/MyDrive/Final
Project/run_outputs/full_scores.csv"   # has *_label, *_conf, *_norm

# OUTPUT
OUT_HEUR = "full_scores_with_heuristics_TUNED.csv"

df = pd.read_csv(FULL_SCORES)

# --- Safety checks ---
req_cols =
{"sent_norm","emot_label","emot_conf","bias_norm","cred_norm","news_id
"}
```

```python
missing = [c for c in req_cols if c not in df.columns]
if missing:
    raise ValueError(f"Missing columns in {FULL_SCORES}: {missing}")

# --- Sentiment heuristic ---
# If sent_norm appears in [0,1], use center=0.5 with neutral band
0.35–0.65.
# If it appears in [-1,1], use neutral band (-0.2 .. +0.2).
def heur_sentiment(x):
    if pd.isna(x):
        return None
    try:
        x = float(x)
    except:
        return None
    if 0.0 <= x <= 1.0:
        if x > 0.65: return "positive"
        if x < 0.35: return "negative"
        return "neutral"
    else:
        if x > 0.20: return "positive"
        if x < -0.20: return "negative"
        return "neutral"

df["heur_sent"] = df["sent_norm"].apply(heur_sentiment)

# --- Emotion heuristic ---
# Only accept non-neutral emotion if emot_conf > 0.60; otherwise mark
Neutral
df["heur_emot"] = np.where(df["emot_conf"] > 0.60,
df["emot_label"].str.lower(), "neutral")

# --- Bias heuristic ---
# Conservative: biased if bias_norm >= 0.66, else neutral
df["heur_bias"] = np.where(df["bias_norm"] >= 0.66, "biased",
"neutral")

# --- Credibility heuristic ---
# Flip towards High if cred_norm >= 0.70, else Low
df["heur_cred"] = np.where(df["cred_norm"] >= 0.70, "high
credibility", "low credibility")

# Save
df.to_csv(OUT_HEUR, index=False)
print(f"□ Heuristic labels added → {OUT_HEUR}")
```

```
□ Heuristic labels added → full_scores_with_heuristics_TUNED.csv
```

```python
import pandas as pd
import numpy as np
```

```
import re
from collections import Counter
from sklearn.metrics import classification_report, confusion_matrix,
cohen_kappa_score, accuracy_score, f1_score

# INPUTS
MANUAL_100 = "/content/drive/MyDrive/Final
Project/manual_label_set_100_labeled.csv"      # gold labels (100
rows)
HEUR_FULL  = "full_scores_with_heuristics_TUNED.csv" # produced by
Cell 1

man = pd.read_csv(MANUAL_100)
heur = pd.read_csv(HEUR_FULL)

df =
man.merge(heur[["news_id","heur_sent","heur_emot","heur_bias","heur_cr
ed"]], on="news_id", how="left")
print("Merged rows:", len(df))

def norm(x):
    if x is None or (isinstance(x, float) and np.isnan(x)): return
None
    s = str(x).strip().lower()
    s = re.sub(r"\s+", " ", s)
    mapping = {
        "pos":"positive","neg":"negative","neu":"neutral",
        "high":"high credibility","low":"low credibility"
    }
    return mapping.get(s, s)

def eval_task(gt_col, pr_col, task_name, labels_order=None, digits=3):
    print("\n" + "="*70)
    print(f"{task_name.upper()} — {gt_col} (manual) vs {pr_col}
(heuristic)")
    sub = df[[gt_col, pr_col]].dropna()
    if sub.empty:
        print("No rows to evaluate.");
        return
    y_true = sub[gt_col].map(norm).tolist()
    y_pred = sub[pr_col].map(norm).tolist()
    labels = labels_order or sorted(set(y_true) | set(y_pred))

    print("\nCounts:")
    print("  Ground truth:", Counter(y_true))
    print("  Predictions :", Counter(y_pred))

    print("\nClassification report (macro):")
    print(classification_report(y_true, y_pred, labels=labels,
zero_division=0, digits=digits))
```

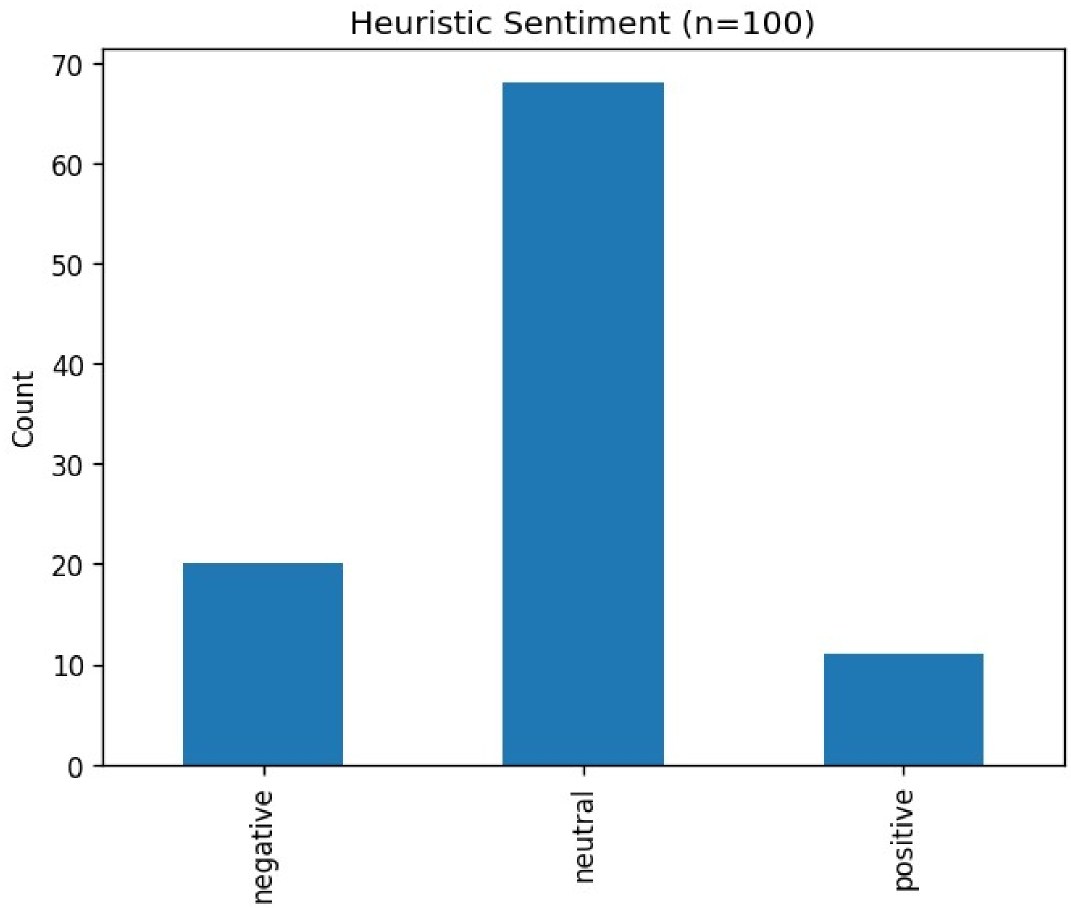
```
    acc   = accuracy_score(y_true, y_pred)
    f1m   = f1_score(y_true, y_pred, average="macro", zero_division=0)
    kappa = cohen_kappa_score(y_true, y_pred)
    print(f"Cohen's κ: {kappa:.3f} | Accuracy: {acc:.3f} | Macro-F1:
{f1m:.3f}")

    cm = confusion_matrix(y_true, y_pred, labels=labels)
    print("\nLabels order:", labels)
    print("Confusion matrix (rows=GT, cols=Pred):")
    print(cm)

# Run all four tasks
eval_task("man_sentiment",   "heur_sent",  "Sentiment",
labels_order=["negative","neutral","positive"])
eval_task("man_emotion",     "heur_emot",  "Emotion",
labels_order=["anger","fear","joy","sadness","surprise","neutral"])
eval_task("man_bias",        "heur_bias",  "Bias",
labels_order=["neutral","biased"])
eval_task("man_credibility", "heur_cred",  "Credibility",
labels_order=["low credibility","high credibility"])
```

```
Merged rows: 99

======================================================================
SENTIMENT — man_sentiment (manual) vs heur_sent (heuristic)

Counts:
  Ground truth: Counter({'neutral': 74, 'negative': 23, 'positive':
2})
  Predictions : Counter({'neutral': 68, 'negative': 20, 'positive':
11})

Classification report (macro):
              precision    recall  f1-score   support

    negative      0.450     0.391     0.419        23
     neutral      0.779     0.716     0.746        74
    positive      0.091     0.500     0.154         2

    accuracy                          0.636        99
   macro avg      0.440     0.536     0.440        99
weighted avg      0.689     0.636     0.658        99

Cohen's κ: 0.169 | Accuracy: 0.636 | Macro-F1: 0.440

Labels order: ['negative', 'neutral', 'positive']
Confusion matrix (rows=GT, cols=Pred):
[[ 9 14  0]
 [11 53 10]
```

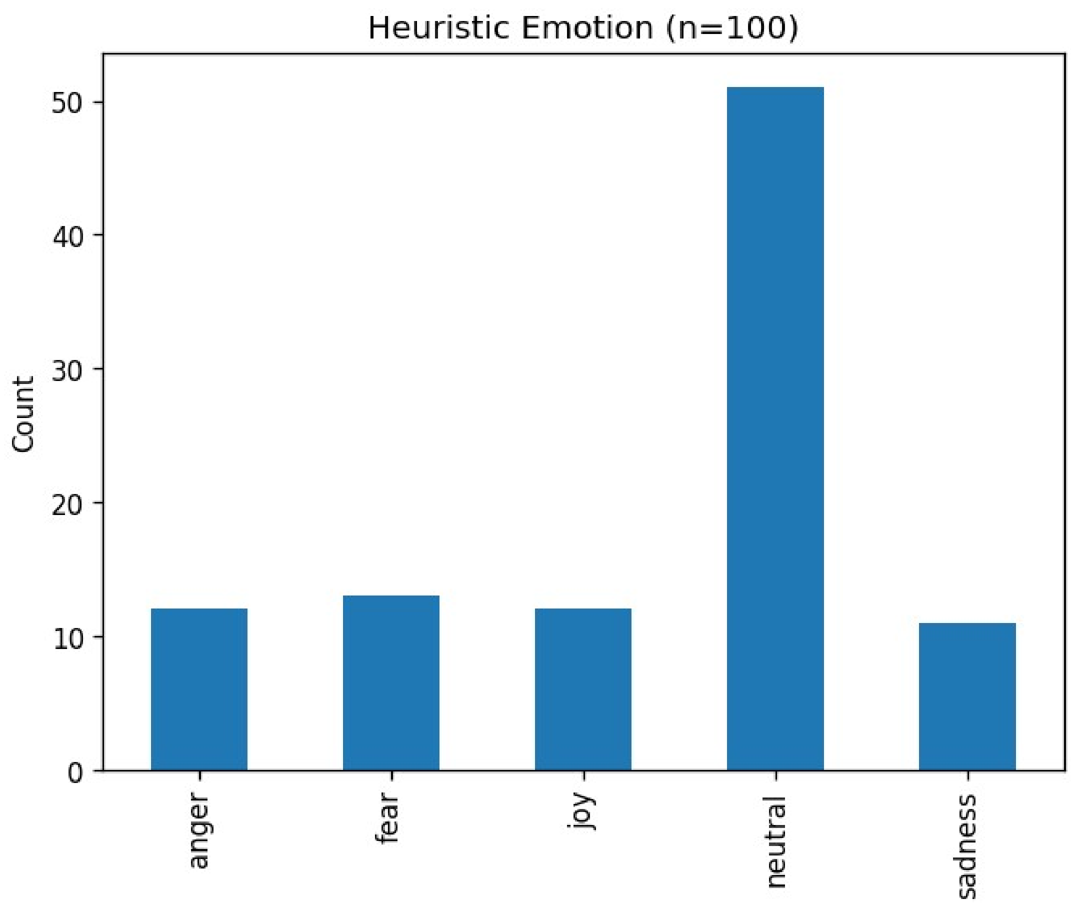
```
 [ 0  1  1]]

======================================================================
EMOTION — man_emotion (manual) vs heur_emot (heuristic)

Counts:
  Ground truth: Counter({'neutral': 66, 'fear': 18, 'anger': 10,
'joy': 4, 'sadness': 1})
  Predictions : Counter({'neutral': 51, 'fear': 13, 'anger': 12,
'joy': 12, 'sadness': 11})

Classification report (macro):
              precision    recall  f1-score   support

       anger      0.333     0.400     0.364        10
        fear      0.231     0.167     0.194        18
         joy      0.167     0.500     0.250         4
     sadness      0.000     0.000     0.000         1
    surprise      0.000     0.000     0.000         0
     neutral      0.725     0.561     0.632        66

    accuracy                          0.465        99
   macro avg      0.243     0.271     0.240        99
weighted avg      0.566     0.465     0.504        99

Cohen's κ: 0.129 | Accuracy: 0.465 | Macro-F1: 0.288

Labels order: ['anger', 'fear', 'joy', 'sadness', 'surprise',
'neutral']
Confusion matrix (rows=GT, cols=Pred):
[[ 4  2  0  0  0  4]
 [ 7  3  0  1  0  7]
 [ 0  0  2  0  0  2]
 [ 0  0  0  0  0  1]
 [ 0  0  0  0  0  0]
 [ 1  8 10 10  0 37]]

======================================================================
BIAS — man_bias (manual) vs heur_bias (heuristic)

Counts:
  Ground truth: Counter({'neutral': 85, 'biased': 14})
  Predictions : Counter({'neutral': 99})

Classification report (macro):
              precision    recall  f1-score   support

     neutral      0.859     1.000     0.924        85
      biased      0.000     0.000     0.000        14
```

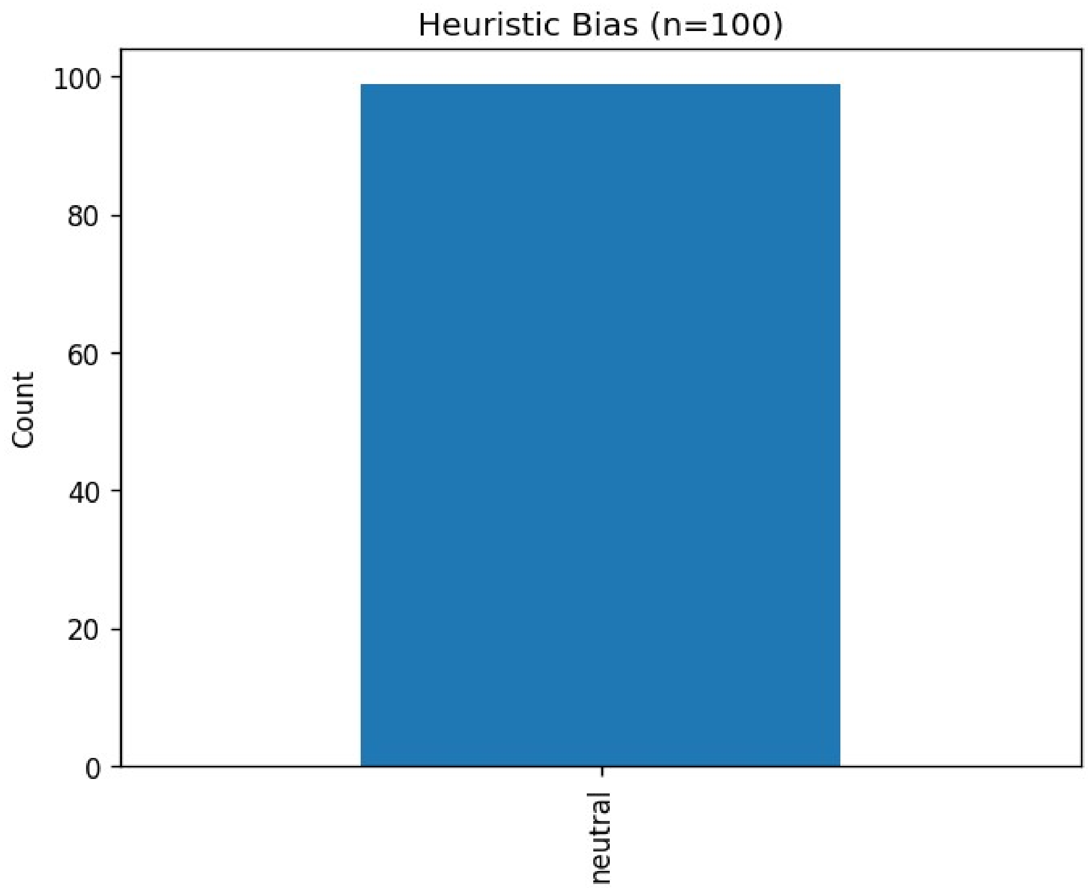
```
    accuracy                           0.859        99
   macro avg      0.429     0.500     0.462        99
weighted avg      0.737     0.859     0.793        99

Cohen's κ: 0.000 | Accuracy: 0.859 | Macro-F1: 0.462

Labels order: ['neutral', 'biased']
Confusion matrix (rows=GT, cols=Pred):
[[85  0]
 [14  0]]

======================================================================
CREDIBILITY — man_credibility (manual) vs heur_cred (heuristic)

Counts:
  Ground truth: Counter({'high credibility': 96, 'low credibility':
3})
  Predictions : Counter({'low credibility': 96, 'high credibility':
3})

Classification report (macro):
                  precision    recall  f1-score   support

 low credibility      0.031     1.000     0.061         3
high credibility      1.000     0.031     0.061        96

        accuracy                          0.061        99
       macro avg      0.516     0.516     0.061        99
    weighted avg      0.971     0.061     0.061        99

Cohen's κ: 0.002 | Accuracy: 0.061 | Macro-F1: 0.061

Labels order: ['low credibility', 'high credibility']
Confusion matrix (rows=GT, cols=Pred):
[[ 3  0]
 [93  3]]
```

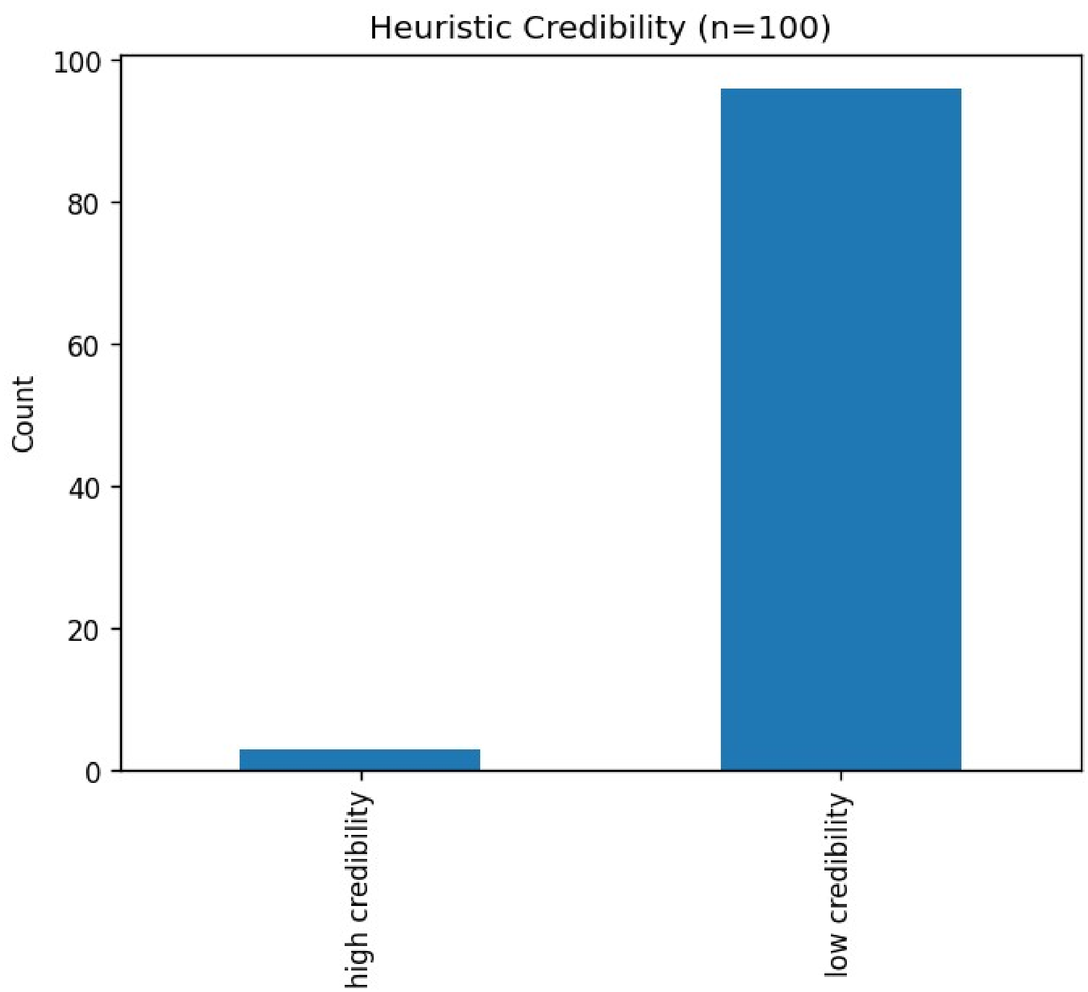
```
import matplotlib.pyplot as plt

def plot_dist(series, title):
    s = series.dropna().map(lambda x: str(x).lower())
    s.value_counts().sort_index().plot(kind="bar")
    plt.title(title); plt.xlabel(""); plt.ylabel("Count"); plt.show()

plot_dist(df["heur_sent"], "Heuristic Sentiment (n=100)")
plot_dist(df["heur_emot"], "Heuristic Emotion (n=100)")
plot_dist(df["heur_bias"], "Heuristic Bias (n=100)")
plot_dist(df["heur_cred"], "Heuristic Credibility (n=100)")
```



Heuristic Sentiment (n=100)
Count
70
60
50
40
30
20
10
0
negative
neutral
positive


Heuristic Emotion (n=100)
Count
0
10
20
30
40
50
anger
fear
joy
neutral
sadness


Heuristic Bias (n=100)
Count
100
80
60
40
20
0
neutral


Heuristic Credibility (n=100)
Count
0
20
40
60
80
100
high credibility
low credibility