LAMPIRAN A

Lampiran 1

Dokumentasi Bersama Responden Ahli

LAMPIRAN B

Code Arimax model

```
import os
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_absolute_error,
mean_squared_error
from sklearn.preprocessing import StandardScaler

warnings.filterwarnings("ignore")
np.random.seed(42)
os.environ["PYTHONHASHSEED"] = "42"

BASE_DIR = Path(r"C:\Users\LENOVO\my-scraper")
LOCKED_DIR = BASE_DIR / "prediksi_locked"
LOCKED_DIR.mkdir(parents=True, exist_ok=True)
NEXT_WEEK_FILE = LOCKED_DIR / "next_week.csv"

ARIMAX_ORDER = (0, 1, 1)
ARIMA_ORDER = (0, 1, 1)

KATEGORI = {
"avg_3_9": ("Umur 3-9 Tahun", "blue"),
"10_20": ("Umur 10-20 Tahun", "green"),
"avg_21_25": ("Umur 21-25 Tahun", "red"),
}

def load_locked(col):
path = LOCKED_DIR / f"locked_{col}.csv"
if path.exists():
lk = pd.read_csv(path, parse_dates=["date"])
lk["date"] = pd.to_datetime(lk["date"])
return lk
return pd.DataFrame(columns=["date", "aktual", "pred_arimax",
"pred_arima"])

def save_locked(col, df_locked):
path = LOCKED_DIR / f"locked_{col}.csv"
df_locked =
df_locked.sort_values("date").drop_duplicates(subset=["date"])
df_locked.to_csv(path, index=False)
```

```
def load_next_week():
if NEXT_WEEK_FILE.exists():
return pd.read_csv(NEXT_WEEK_FILE)
return pd.DataFrame()

def save_next_week(df_nw):
df_nw = df_nw.sort_values(["next_date", "col"]).drop_duplicates(
subset=["col", "next_date"], keep="first"
)
df_nw.to_csv(NEXT_WEEK_FILE, index=False)

def hitung_metrik(y_true, y_pred):
y_true, y_pred = np.asarray(y_true, float), np.asarray(y_pred,
float)
mae = mean_absolute_error(y_true, y_pred)
rmse = np.sqrt(mean_squared_error(y_true, y_pred))
mape = np.mean(np.abs((y_true - y_pred) / y_true)) * 100
return mae, rmse, mape

df = pd.read_csv(BASE_DIR / "dataset_arimax.csv")
df["date"] = pd.to_datetime(df["date"])
df = df.sort_values("date").set_index("date")

df["bull_l2"] = df["bullish_ratio"].shift(2)
df["bear_l1"] = df["bearish_ratio"].shift(1)
EXOG_COLS = ["bull_l2", "bear_l1"]

df = df.dropna().reset_index(drop=False).set_index("date")

hasil_semua = {}
next_week_rows = []
fig, axes = plt.subplots(3, 1, figsize=(14, 16), sharex=False)
df_nw_existing = load_next_week()

for idx, (col, (nama, warna)) in enumerate(KATEGORI.items()):

y = df[col]
X_best = df[EXOG_COLS]

split = int(len(df) * 0.8)
y_train, y_test = y[:split], y[split:]
X_train_best, X_test_best = X_best[:split], X_best[split:]

scaler = StandardScaler()
X_train_sc = pd.DataFrame(
scaler.fit_transform(X_train_best),
```

```
columns=X_train_best.columns, index=X_train_best.index
)
X_test_sc = pd.DataFrame(
scaler.transform(X_test_best),
columns=X_test_best.columns, index=X_test_best.index
)
X_full_sc = pd.DataFrame(
scaler.transform(X_best),
columns=X_best.columns, index=X_best.index
)

n_test = len(y_test)

df_locked = load_locked(col)
locked_dates =
set(pd.to_datetime(df_locked["date"]).dt.normalize())
test_dates = pd.to_datetime(y_test.index).normalize()
new_mask = ~pd.Index(test_dates).isin(locked_dates)

new_rows = []
if sum(new_mask) > 0:
history_y = list(y_train.values)
history_X = list(X_train_sc.values)

if not df_locked.empty:
for _, lrow in df_locked.sort_values("date").iterrows():
ldate = pd.Timestamp(lrow["date"])
if ldate in y_test.index:
i_in = list(y_test.index).index(ldate)
history_y.append(float(lrow["aktual"]))
history_X.append(X_test_sc.values[i_in])

for i in range(n_test):
test_date = y_test.index[i]
t_norm = pd.Timestamp(test_date).normalize()

if t_norm in locked_dates:
continue

np.random.seed(42)
try:
m_x = SARIMAX(history_y, exog=history_X, order=ARIMAX_ORDER).fit(
disp=False, method="powell"
)
yhat = float(m_x.forecast(steps=1,
exog=X_test_sc.iloc[[i]].values)[0])
except Exception:
```

```
yhat = float(y_test.iloc[i])

try:
m_a = ARIMA(history_y, order=ARIMA_ORDER).fit()
arima_yhat = float(m_a.forecast(steps=1)[0])
except Exception:
arima_yhat = float(y_test.iloc[i])

new_rows.append({
"date": test_date,
"aktual": float(y_test.iloc[i]),
"pred_arimax": round(yhat, 2),
"pred_arima": round(arima_yhat, 2),
})

history_y.append(float(y_test.iloc[i]))
history_X.append(X_test_sc.values[i])

new_df = pd.DataFrame(new_rows)
new_df["date"] = pd.to_datetime(new_df["date"])
df_locked = pd.concat([df_locked, new_df], ignore_index=True)
save_locked(col, df_locked)

df_ls =
df_locked.sort_values("date").drop_duplicates(subset=["date"])
mask_t = pd.to_datetime(df_ls["date"]).dt.normalize().isin(
set(pd.to_datetime(y_test.index).normalize())
)
df_tl = df_ls[mask_t].copy()
df_tl.index = pd.to_datetime(df_tl["date"])
df_tl = df_tl.sort_index()

pred_arimax = pd.Series(df_tl["pred_arimax"].values.astype(float),
index=df_tl.index)
pred_arima = pd.Series(df_tl["pred_arima"].values.astype(float),
index=df_tl.index)
aktual_test = pd.Series(df_tl["aktual"].values.astype(float),
index=df_tl.index)

valid = aktual_test.notna() & pred_arimax.notna() &
pred_arima.notna()
mae_x, rmse_x, mape_x = hitung_metrik(aktual_test[valid],
pred_arimax[valid])
mae_a, rmse_a, mape_a = hitung_metrik(aktual_test[valid],
pred_arima[valid])

next_date = y.index[-1] + pd.Timedelta(weeks=1)
```

```
next_date_str = next_date.date().isoformat()

already_in_nw = False
if not df_nw_existing.empty and "col" in df_nw_existing.columns:
already_in_nw = (
(df_nw_existing["col"] == col) &
(df_nw_existing["next_date"].astype(str) == next_date_str)
).any()

if not already_in_nw:
X_full_arr = X_full_sc.values.copy()
exog_future = X_full_arr[[-1]]

np.random.seed(42)
fit_x = SARIMAX(y.values, exog=X_full_arr,
order=ARIMAX_ORDER).fit(
disp=False, method="powell"
)
fit_a = ARIMA(y.values, order=ARIMA_ORDER).fit()

next_arimax = float(fit_x.forecast(steps=1, exog=exog_future)[0])
next_arima = float(fit_a.forecast(steps=1)[0])

next_week_rows.append({
"col": col,
"nama": nama,
"next_date": next_date_str,
"last_price": round(float(y.iloc[-1]), 2),
"next_arimax": round(next_arimax, 2),
"selisih_arimax": round(next_arimax - float(y.iloc[-1]), 2),
"next_arima": round(next_arima, 2),
"selisih_arima": round(next_arima - float(y.iloc[-1]), 2),
"mape_arimax": round(mape_x, 4),
"mape_arima": round(mape_a, 4),
"mae_arimax": round(mae_x, 2),
"rmse_arimax": round(rmse_x, 2),
"mae_arima": round(mae_a, 2),
"rmse_arima": round(rmse_a, 2),
"order_arimax": str(ARIMAX_ORDER),
"order_arima": str(ARIMA_ORDER),
"exog_used": "|".join(EXOG_COLS),
})

hasil_semua[col] = {
"nama": nama,
"mae": mae_x, "rmse": rmse_x, "mape": mape_x,
"arima_mape": mape_a, "arima_mae": mae_a, "arima_rmse": rmse_a,
```

```
"y_train": y_train,
"y_test": aktual_test,
"pred": pred_arimax,
"arima_pred": pred_arima,
}

axes[idx].plot(y_train.index, y_train, label="Train",
color="gray", lw=1, alpha=0.7)
axes[idx].plot(aktual_test.index, aktual_test, label="Aktual",
color=warna, lw=1.5)
axes[idx].plot(pred_arimax.index, pred_arimax,
label=f"ARIMAX {ARIMAX_ORDER} (MAPE {mape_x:.2f}%)",
color="red", ls="--", lw=1.5)
axes[idx].plot(pred_arima.index, pred_arima,
label=f"ARIMA {ARIMA_ORDER} (MAPE {mape_a:.2f}%)",
color="orange", ls=":", lw=1.5)
axes[idx].set_title(
f"{nama} | ARIMAX MAPE {mape_x:.2f}% vs ARIMA MAPE {mape_a:.2f}%"
)
axes[idx].set_ylabel("Harga TBS (Rp)")
axes[idx].legend(fontsize=8)
axes[idx].grid(True, alpha=0.3)

if next_week_rows:
df_nw_new = pd.DataFrame(next_week_rows)
df_nw_all = (
pd.concat([df_nw_existing, df_nw_new], ignore_index=True)
if not df_nw_existing.empty else df_nw_new
)
save_next_week(df_nw_all)

for col, h in hasil_semua.items():
selisih = h["arima_mape"] - h["mape"]
status = "ARIMAX menang" if selisih > 0 else "ARIMA menang"
print(
f"{h['nama']:<20} MAE_ARIMAX={h['mae']:.2f}
MAE_ARIMA={h['arima_mae']:.2f} "
f"RMSE_ARIMAX={h['rmse']:.2f} RMSE_ARIMA={h['arima_rmse']:.2f} "
f"MAPE_ARIMAX={h['mape']:.2f}% MAPE_ARIMA={h['arima_mape']:.2f}%
({status})"
)

plt.tight_layout()
plt.savefig(str(BASE_DIR / "arimax_hasil.png"), dpi=150,
bbox_inches="tight")
plt.show()
```

```
for col, h in hasil_semua.items():
pd.DataFrame({
"aktual": h["y_test"].values,
"prediksi_arimax": h["pred"].values,
"prediksi_arima": h["arima_pred"].values,
"error_arimax": h["y_test"].values - h["pred"].values,
"error_arima": h["y_test"].values - h["arima_pred"].values,
}, index=h["y_test"].index).to_csv(
str(BASE_DIR / f"hasil_arimax_{col}_final.csv")
)
```

Code pelabelan Rule Based

```
import pandas as pd
import re
from google.colab import files

print('Upload file CSV...')
uploaded = files.upload()
filename = list(uploaded.keys())[0]
df = pd.read_csv(filename)
print(f'Total berita: {len(df)}')

RULES = {
    'bullish': [
        r'naik rp[\.\s]?\d',
        r'harga cpo.*naik',
        r'harga tbs.*naik',
        r'kpbn.*naik',
        r'bursa malaysia.*naik',
        r'menguat.*ringgit',
        r'ringgit.*menguat',
        r'naik.*ringgit',
        r'ringgit.*naik',
        r'india.*naikkan impor',
        r'china.*naikkan impor',
        r'permintaan.*meningkat',
        r'ekspor.*melonjak',
        r'ekspor.*meningkat',
        r'thailand.*batasi ekspor',
        r'malaysia.*batasi ekspor',
        r'stok.*malaysia.*turun',
        r'pasokan.*ketat',
        r'b40.*resmi.*berlaku',
        r'b50.*resmi.*berlaku',
        r'biodiesel.*resmi.*berlaku',
        r'resmi.*berlaku.*biodiesel',
```

```
        r'mandatori.*biodiesel.*resmi',
    ],
    'bearish': [
        r'turun rp[\.\s]?\d',
        r'harga cpo.*turun',
        r'harga tbs.*turun',
        r'kpbn.*turun',
        r'bursa malaysia.*turun',
        r'melemah.*ringgit',
        r'ringgit.*melemah',
        r'turun.*ringgit',
        r'ringgit.*turun',
        r'india.*pangkas.*impor',
        r'india.*memangkas.*impor',
        r'india.*kurangi.*impor',
        r'china.*kurangi.*impor',
        r'permintaan.*lesu',
        r'ekspor.*turun',
        r'ekspor.*melemah',
        r'stok.*malaysia.*melonjak',
        r'stok.*malaysia.*naik',
        r'oversupply',
        r'kelebihan pasokan',
        r'larangan ekspor.*berlaku',
    ],
    'irrelevant': [
        r'\bbuku\b',
        r'kisah nyata',
        r'biografi',
        r'penghargaan csr',
        r'\bseminar\b',
        r'\bkonferensi\b',
        r'\bworkshop\b',
        r'\bwebinar\b',
        r'beasiswa',
        r'majalah infosawit edisi',
        r'pameran produk',
        r'ajang.*expo',
        r'kunjungan kerja',
        r'program pelatihan',
        r'pelatihan strategis',
        r'otomotif',
        r'gaikindo',
        r'properti',
    ],
    'neutral': [
        r'merujuk hasil dari tim penetapan harga',
```

```
        r'tim penetapan harga tbs',
        r'periode \d+.*\d+.*20\d\d',
        r'sawit umur \d+ tahun rp',
        r'harga referensi.*kemendag',
        r'harga indeks pasar.*biodiesel',
        r'hip.*biodiesel.*april',
        r'laba bersih.*rp.*triliun',
        r'pendapatan.*rp.*triliun',
        r'total aset.*rp.*triliun',
        r'diperkirakan.*harga',
        r'diproyeksikan',
        r'analis.*memperkirakan',
    ]
}

def rule_based_label(text):
    text_lower = str(text).lower()
    scores = {'bullish': 0, 'bearish': 0, 'neutral': 0,
'irrelevant': 0}

    for label, patterns in RULES.items():
        for pattern in patterns:
            if re.search(pattern, text_lower):
                scores[label] += 1

    has_price_rp  = bool(re.search(r'rp[\.\s]?\d{1,3}[\.,]\d{3}',
text_lower))
    has_price_rm  =
bool(re.search(r'rm[\.\s]?\d{1,3}[\.,]\d{3}|ringgit', text_lower))
    has_naik      = bool(re.search(r'\bnaik\b', text_lower))
    has_turun     = bool(re.search(r'\bturun\b|\bmelemah\b',
text_lower))
    has_penetapan = bool(re.search(r'merujuk hasil dari tim
penetapan', text_lower))

    if has_penetapan:
        return 'neutral', 'high'

    if has_price_rp and has_naik and not has_turun:
        scores['bullish'] += 4
    elif has_price_rm and has_naik and not has_turun:
        scores['bullish'] += 3

    if has_price_rp and has_turun and not has_naik:
        scores['bearish'] += 4
    elif has_price_rm and has_turun and not has_naik:
        scores['bearish'] += 3
```

```
    if has_naik and has_turun:
        kpbn_naik  = bool(re.search(r'kpbn.*naik|franco
dumai.*naik|naik rp[\.\s]?\d', text_lower))
        kpbn_turun = bool(re.search(r'kpbn.*turun|franco
dumai.*turun|turun rp[\.\s]?\d', text_lower))
        if kpbn_naik and not kpbn_turun:
            scores['bullish'] += 3
        elif kpbn_turun and not kpbn_naik:
            scores['bearish'] += 3

    max_score = max(scores.values())
    if max_score == 0:
        return 'neutral', 'low'

    best = max(scores, key=scores.get)
    sorted_scores = sorted(scores.values(), reverse=True)
    margin = sorted_scores[0] - sorted_scores[1]
    confidence = 'high' if margin >= 4 else ('medium' if margin >= 2
else 'low')

    return best, confidence

print('Melabeli semua berita...')

labels, confidences, needs_reviews = [], [], []

for text in df['text']:
    label, conf = rule_based_label(text)
    labels.append(label)
    confidences.append(conf)
    needs_reviews.append(conf == 'low')

df['label']        = labels
df['confidence']   = confidences
df['source']       = 'rule_based'
df['needs_review'] = needs_reviews

print(f'Selesai!')
print(f'\nDistribusi label:')
print(df['label'].value_counts())
print(f'\nDistribusi confidence:')
print(df['confidence'].value_counts())
print(f'\nPerlu review manual: {df["needs_review"].sum()} berita
({df["needs_review"].mean()*100:.1f}%)')

OUTPUT = '/content/sawit_rulebased_labeled.csv'
```

```
df.to_csv(OUTPUT, index=False)
files.download(OUTPUT)
print(f'\nFile didownload: {OUTPUT}')
```

Code Pelabelan API Groq

```
!pip install groq pandas tqdm openpyxl -q

import pandas as pd
import os, json, re, time
from collections import Counter
from tqdm import tqdm
from groq import Groq
from google.colab import files, userdata

try:
    api_key = userdata.get('GROQ_API_KEY')
    print('API key dari Colab Secrets')
except:
    api_key =
'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'

client = Groq(api_key=api_key)

test = client.chat.completions.create(
    model='llama-3.3-70b-versatile',
    messages=[{"role": "user", "content": "Balas hanya: OK"}],
    max_tokens=5
)
print(f'Groq aktif: {test.choices[0].message.content}')

SYSTEM_PROMPT = """Kamu adalah trader CPO berpengalaman yang membaca
berita sawit untuk memutuskan posisi beli/jual.
Kamu hanya peduli pada berita yang BENAR-BENAR menggerakkan harga
pasar CPO/TBS secara nyata.

═══════════════════════════════════════════
PANDUAN MEMBACA BERITA KPBN (sangat umum di dataset ini):
═══════════════════════════════════════════
Berita KPBN selalu punya dua bagian: harga domestik Indonesia (KPBN)
dan harga Bursa Malaysia.
PRIORITASKAN harga KPBN Franco Dumai sebagai sinyal utama:
- Jika kalimat pembuka KPBN menyebut "naik Rp X/kg" → BULLISH
- Jika kalimat pembuka KPBN menyebut "turun Rp X/kg" → BEARISH
- Jika kalimat pembuka KPBN menyebut "Withdraw/WD" (tidak ada
transaksi) → lihat harga Bursa Malaysia
  * Bursa Malaysia naik → BULLISH
```

```
  * Bursa Malaysia turun → BEARISH
  * Tidak jelas → NEUTRAL
- Jika KPBN naik/turun TIPIS (di bawah Rp100/kg) DAN arah Bursa
Malaysia BERLAWANAN →
  ini sinyal campuran, prioritaskan tetap KPBN tapi turunkan
confidence jadi "low".
Abaikan kata "naik/turun" di paragraf lain jika sudah ada sinyal
KPBN yang jelas di awal.

═══════════════════════════════════════════
BATAS NEUTRAL vs IRRELEVANT (sering tertukar, perhatikan baik-baik):
═══════════════════════════════════════════
NEUTRAL = topik MASIH dalam ranah sawit/CPO/kebijakan terkait, tapi
TIDAK ada sinyal harga konkret hari ini.
  Termasuk: penertiban sawit ilegal, regulasi tata kelola sawit, isu
lingkungan/sawit, statistik rutin,
  penetapan harga TBS dinas provinsi, HR CPO Kemendag, HIP biodiesel
ESDM, laporan keuangan perusahaan sawit.

IRRELEVANT = topik SAMA SEKALI DI LUAR sawit/CPO sebagai komoditas.
  Termasuk: buku/resensi buku, profil tokoh, penghargaan CSR,
pameran/event edukasi non-harga,
  seminar/konferensi/pelatihan, berita otomotif/properti/sektor
lain.

Aturan cepat: kalau topiknya "sawit melakukan X" tapi X tidak
berkaitan harga → NEUTRAL.
Kalau topiknya "X terjadi DI DEKAT/SEKITAR sawit" tapi X-nya sendiri
bukan soal sawit (mis. buku, seremoni) → IRRELEVANT.

═══════════════════════════════════════════
ATURAN LABEL ARAH:
═══════════════════════════════════════════
BULLISH — hanya jika ada salah satu sinyal nyata:
  ▸ KPBN Franco Dumai naik dengan angka rupiah konkret hari ini
  ▸ Bursa Malaysia naik dengan angka ringgit konkret hari ini
  ▸ Kebijakan biodiesel RESMI BERLAKU mulai tanggal tertentu (bukan
"direncanakan")
  ▸ Capaian/realisasi program biodiesel yang dilaporkan
POSITIF/meningkat (meski ada catatan tantangan
    teknis di paragraf lain, selama narasi UTAMA tetap capaian
positif → tetap BULLISH)
  ▸ India/China KONFIRMASI naikkan volume impor CPO hari ini
  ▸ Malaysia/Thailand RESMI batasi ekspor CPO efektif sekarang
  ▸ Stok CPO Malaysia turun di bawah ekspektasi (data MPOB resmi)

BEARISH — hanya jika ada salah satu sinyal nyata:
```

```
  ▸ KPBN Franco Dumai turun dengan angka rupiah konkret hari ini
  ▸ Bursa Malaysia turun dengan angka ringgit konkret hari ini
  ▸ India/China KONFIRMASI pangkas atau batalkan impor CPO hari ini
  ▸ Stok CPO Malaysia melonjak di atas ekspektasi (data MPOB resmi)
  ▸ Larangan ekspor CPO Indonesia RESMI diberlakukan

═══════════════════════════════════════════
CONTOH KASUS SULIT (pelajari pola ini):
═══════════════════════════════════════════
1. "Capaian kinerja program biodiesel B50 meningkat, meski ada
tantangan ketergantungan impor metanol"
   → BULLISH (narasi utama = capaian positif; tantangan teknis hanya
catatan sampingan, bukan sinyal pembalik arah)
2. "Kemenhut-Gakkum tertibkan sawit ilegal"
   → NEUTRAL (bukan irrelevant; ini isu regulasi/tata kelola sawit,
tapi tidak ada dampak harga langsung hari ini)
3. "Resensi buku [judul], profil tokoh, atau event edukasi tanpa
angka harga"
   → IRRELEVANT (topik di luar pergerakan harga CPO/sawit sebagai
komoditas)
4. "KPBN turun Rp90/kg, sementara Bursa Malaysia menguat"
   → BEARISH dengan confidence LOW (sinyal campuran; KPBN tetap
prioritas tapi turunkan keyakinan)
5. "KPBN naik Rp19/kg ke harga tertentu, tanpa konteks tambahan
lain"
   → BULLISH (sinyal KPBN positif walau kecil, tetap dihitung
sebagai sinyal harga nyata)

═══════════════════════════════════════════
CARA MENJAWAB:
═══════════════════════════════════════════
Sebelum menjawab, pikirkan singkat dalam hati (jangan ditulis):
Step 1: Apakah topik ini dalam ranah sawit/CPO sama sekali? Jika
tidak → IRRELEVANT.
Step 2: Jika ya, apakah ada sinyal harga konkret hari ini
(KPBN/Bursa Malaysia/kebijakan resmi/data MPOB)?
        Jika tidak ada → NEUTRAL.
Step 3: Jika ada, tentukan arah BULLISH atau BEARISH sesuai aturan
di atas.
Step 4: Jika sinyal bercampur/berlawanan arah, tetap putuskan satu
arah dominan, confidence = low.

Tanyakan ke diri sendiri: "Apakah berita ini membuatku MENGUBAH
POSISI TRADING hari ini?"
Jika tidak yakin antara neutral/irrelevant → pilih NEUTRAL (karena
masih dalam ranah sawit).
```

```
Balas HANYA JSON ini (tanpa penjelasan, tanpa reasoning tertulis):
{"label": "bullish|bearish|neutral|irrelevant", "confidence":
"high|medium|low"}"""

KEYWORDS = {
    'bullish': [
        r'naik rp[\.\s]?\d', r'harga cpo.*naik', r'kpbn.*naik',
        'bursa malaysia.*naik', 'menguat', 'biodiesel.*resmi',
        'stok.*turun', 'ekspor melonjak',
    ],
    'bearish': [
        r'turun rp[\.\s]?\d', r'harga cpo.*turun', r'kpbn.*turun',
        'bursa malaysia.*turun', 'melemah', 'oversupply',
        'india.*pangkas', 'india.*memangkas',
    ],
    'irrelevant': [
        'buku', 'novel', 'biografi', 'penghargaan csr',
        'seminar', 'konferensi', 'workshop', 'beasiswa',
        'majalah infosawit edisi',
    ]
}

def rule_based_label(text):
    text_lower = str(text).lower()
    scores = {'bullish': 0, 'bearish': 0, 'irrelevant': 0,
'neutral': 0}
    for label, keywords in KEYWORDS.items():
        for kw in keywords:
            if re.search(kw, text_lower):
                scores[label] += 1
    has_price = bool(re.search(r'rp[\.\s]?\d{1,3}[\.,]\d{3}',
text_lower))
    if has_price and 'naik' in text_lower:
        scores['bullish'] += 3
    if has_price and ('turun' in text_lower or 'melemah' in
text_lower):
        scores['bearish'] += 3
    max_score = max(scores.values())
    if max_score == 0:
        return 'neutral', 'low'
    best = max(scores, key=scores.get)
    sorted_scores = sorted(scores.values(), reverse=True)
    margin = sorted_scores[0] - sorted_scores[1]
    conf = 'high' if margin >= 3 else ('medium' if margin >= 2 else
'low')
    return best, conf
```

```
VALID_LABELS = ['bullish', 'bearish', 'neutral', 'irrelevant']

def _single_groq_call(text, client, max_chars, temperature):
    response = client.chat.completions.create(
        model='llama-3.3-70b-versatile',
        messages=[
            {"role": "system", "content": SYSTEM_PROMPT},
            {"role": "user", "content": str(text)[:max_chars]}
        ],
        max_tokens=60,
        temperature=temperature
    )
    raw = response.choices[0].message.content.strip()
    raw = raw.replace('```json', '').replace('```', '').strip()
    result = json.loads(raw)
    label = result.get('label', 'neutral').lower().strip()
    conf  = result.get('confidence', 'medium').lower().strip()
    if '|' in label or label not in VALID_LABELS:
        found = False
        for v in VALID_LABELS:
            if v in label:
                label = v
                found = True
                break
        if not found:
            raise ValueError(f'Label tidak valid: {label}')
    if conf not in ['high', 'medium', 'low']:
        conf = 'medium'
    return label, conf

def label_with_groq(text, client, max_chars=1200,
use_majority_vote=False, n_votes=3):
    try:
        if not use_majority_vote:
            label, conf = _single_groq_call(text, client, max_chars,
temperature=0.1)
            return label, conf, 'groq'

        labels, confs = [], []
        for temp in [0.1, 0.4, 0.7]:
            try:
                l, c = _single_groq_call(text, client, max_chars,
temperature=temp)
                labels.append(l)
                confs.append(c)
            except Exception:
                continue
```

```
            time.sleep(1)

        if not labels:
            raise ValueError('Semua vote gagal')

        vote = Counter(labels).most_common(1)[0]
        final_label, vote_count = vote
        if vote_count == len(labels):
            final_conf = 'high'
        elif vote_count >= 2:
            final_conf = 'medium'
        else:
            final_conf = 'low'
        return final_label, final_conf, 'groq_majority'

    except json.JSONDecodeError:
        label, conf = rule_based_label(text)
        return label, conf, 'fallback_parseerror'
    except Exception as e:
        err = str(e)
        if 'rate_limit' in err.lower() or '429' in err:
            time.sleep(10)
            label, conf = rule_based_label(text)
            return label, conf, 'fallback_ratelimit'
        label, conf = rule_based_label(text)
        return label, conf, 'fallback_error'

CHECKPOINT = '/content/groq_full_checkpoint.csv'
if os.path.exists(CHECKPOINT):
    os.remove(CHECKPOINT)
    print('Checkpoint lama dihapus')

print('\nUpload file Excel (.xlsx)...')
uploaded = files.upload()
filename = list(uploaded.keys())[0]

df_raw = pd.read_excel(filename, header=None)

df_raw['combined'] = df_raw.astype(str).agg(','.join, axis=1)
split_result = df_raw['combined'].str.split(',', n=1, expand=True)

df = pd.DataFrame({
    'date': split_result[0],
    'text': split_result[1]
})
df['text'] = df['text'].str.strip().str.strip('"')
```

```
df['date'] = pd.to_datetime(df['date'], errors='coerce')
df = df.dropna(subset=['text']).reset_index(drop=True)
print(f'Total berita: {len(df)}')
print(df.head())

NUM_WORKERS = 1
DELAY       = 2.5
SAVE_EVERY  = 100

USE_MAJORITY_VOTE = True

df_out = df.copy()
df_out['ai_label']      = None
df_out['ai_confidence'] = None
df_out['ai_source']     = None
df_out['needs_review']  = None

todo = df_out.index.tolist()
print(f'Sisa: {len(todo)} berita')
print(f'Mode: {"Majority Vote (3x call)" if USE_MAJORITY_VOTE else
"Single Call"}')
print(f'Estimasi: {len(todo)*DELAY*(3 if USE_MAJORITY_VOTE else
1)/60:.0f} menit\n')

groq_count = fallback_count = 0
start_time = time.time()

for i, idx in enumerate(tqdm(todo, desc='Labeling')):
    text = df_out.loc[idx, 'text']
    label, conf, source = label_with_groq(text, client,
use_majority_vote=USE_MAJORITY_VOTE)

    df_out.loc[idx, 'ai_label']      = label
    df_out.loc[idx, 'ai_confidence'] = conf
    df_out.loc[idx, 'ai_source']     = source
    df_out.loc[idx, 'needs_review']  = (conf == 'low' or 'fallback'
in source)

    if 'groq' in source:
        groq_count += 1
    else:
        fallback_count += 1

    if (i + 1) % SAVE_EVERY == 0:
        df_out.to_csv(CHECKPOINT, index=False)
        elapsed = (time.time() - start_time) / 60
        remaining = (len(todo) - (i+1)) * DELAY / 60
```

```
        groq_pct = groq_count / (i+1) * 100
        print(f'[{i+1}/{len(todo)}] Groq: {groq_count}
({groq_pct:.0f}%) | '
              f'Fallback: {fallback_count} | '
              f'Elapsed: {elapsed:.1f}m | ETA: {remaining:.0f}m')

    time.sleep(DELAY)

df_out.to_csv(CHECKPOINT, index=False)
elapsed_total = (time.time() - start_time) / 60
print(f'\nSELESAI dalam {elapsed_total:.1f} menit!')
print(f'Groq: {groq_count} | Fallback: {fallback_count}')
print('\nDistribusi label:')
print(df_out['ai_label'].value_counts())

print('\n=== Ringkasan per Artikel ===')
for i, row in df_out.iterrows():
    print(f"{i+1}. [{row['ai_label'].upper()}] (conf:
{row['ai_confidence']}) — {str(row['text'])[:80]}...")
```

Code Fine-Tuning Indobert

```
!pip install transformers datasets scikit-learn torch -q

import pandas as pd
import csv
import io
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from transformers import (AutoTokenizer,
AutoModelForSequenceClassification,
                          get_linear_schedule_with_warmup)
from torch.optim import AdamW
from sklearn.model_selection import train_test_split
from sklearn.metrics import (accuracy_score, f1_score,
                             classification_report,
confusion_matrix)
from google.colab import files
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

print('Upload berita.csv...')
uploaded = files.upload()
```

```
filename = list(uploaded.keys())[0]
raw_data = uploaded[filename].decode('utf-8').splitlines()

LABEL_VALID      = {'bullish', 'bearish', 'neutral', 'irrelevant'}
CONFIDENCE_VALID = {'high', 'medium', 'low'}

parsed_rows = []
skip_header  = True

for line in raw_data:
    if not line.strip():
        continue

    if skip_header and line.lower().startswith('date'):
        skip_header = False
        continue

    stripped = line
    if stripped.startswith('"') and stripped.endswith('"'):
        stripped = stripped[1:-1]

    try:
        row = next(csv.reader([stripped], quotechar='"',
doublequote=True))
    except Exception:
        continue

    if len(row) < 4:
        continue

    date       = row[0].strip().strip('"')
    confidence = row[-1].strip().strip('"').lower()
    sentiment  = row[-2].strip().strip('"').lower()

    text = ', '.join(row[1:-2]).strip().strip('"')

    if sentiment not in LABEL_VALID:
        continue
    if confidence not in CONFIDENCE_VALID:
        continue

    parsed_rows.append({
        'date'      : date,
        'text'      : text,
        'sentiment' : sentiment,
        'confidence': confidence,
    })
```

```
df_clean = pd.DataFrame(parsed_rows)

print(f'\nTotal baris mentah (tanpa header): {len(raw_data) - 1}')
print(f'Berhasil di-parse dengan label valid: {len(df_clean)}')
print(f'\nDistribusi sentiment:')
print(df_clean['sentiment'].value_counts())
print(f'\nDistribusi confidence:')
print(df_clean['confidence'].value_counts())

df_clean = df_clean[df_clean['confidence'].isin(['high',
'medium'])].copy()
print(f'\nSetelah filter high/medium confidence: {len(df_clean)}
baris')
print(df_clean['sentiment'].value_counts())

if len(df_clean) < 50:
    raise ValueError(
        f"Data terlalu sedikit ({len(df_clean)} baris). "
        "Cek apakah format CSV sudah benar dan label sentimen
valid."
    )

label_names = ['bullish', 'bearish', 'neutral', 'irrelevant']
label2id    = {l: i for i, l in enumerate(label_names)}
id2label    = {i: l for l, i in label2id.items()}

df_clean = df_clean[df_clean['sentiment'].isin(label_names)].copy()
df_clean['label_id'] = df_clean['sentiment'].map(label2id)

print(f'\nDistribusi label final untuk training:')
print(df_clean['sentiment'].value_counts())

train_df, temp_df = train_test_split(
    df_clean, test_size=0.3, random_state=42,
stratify=df_clean['sentiment']
)
val_df, test_df = train_test_split(
    temp_df, test_size=0.5, random_state=42,
stratify=temp_df['sentiment']
)

print(f'\nSplit:')
print(f'  Train : {len(train_df)}')
print(f'  Val   : {len(val_df)}')
print(f'  Test  : {len(test_df)}')
```

```
class_counts         =
train_df['label_id'].value_counts().sort_index().values
total_samples        = sum(class_counts)
class_weights        = total_samples / (len(label_names) *
class_counts)
device               = torch.device('cuda' if
torch.cuda.is_available() else 'cpu')
class_weights_tensor = torch.tensor(class_weights,
dtype=torch.float).to(device)
print(f'\nDevice  : {device}')
print(f'Weights : {dict(zip(label_names, class_weights.round(3)))}')

MODEL_NAME = 'cahya/bert-base-indonesian-1.5G'
MAX_LEN    = 256
BATCH_SIZE = 16
EPOCHS     = 8
LR         = 2e-5

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME, use_fast=True)

class SawitDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len):
        self.texts     = [str(t)[:1000] for t in texts]
        self.labels    = labels
        self.tokenizer = tokenizer
        self.max_len   = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        enc = self.tokenizer(
            self.texts[idx],
            max_length=self.max_len,
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )
        return {
            'input_ids'     : enc['input_ids'].squeeze(),
            'attention_mask': enc['attention_mask'].squeeze(),
            'label'         : torch.tensor(int(self.labels[idx]),
dtype=torch.long)
        }

train_dataset = SawitDataset(train_df['text'].values,
train_df['label_id'].values, tokenizer, MAX_LEN)
```

```
val_dataset   =
SawitDataset(val_df['text'].values,   val_df['label_id'].values,   t
okenizer, MAX_LEN)
test_dataset  =
SawitDataset(test_df['text'].values,  test_df['label_id'].values,  t
okenizer, MAX_LEN)

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True)
val_loader   = DataLoader(val_dataset,   batch_size=BATCH_SIZE,
shuffle=False)
test_loader  = DataLoader(test_dataset,  batch_size=BATCH_SIZE,
shuffle=False)

model = AutoModelForSequenceClassification.from_pretrained(
    MODEL_NAME,
    num_labels=4,
    id2label=id2label,
    label2id=label2id,
    ignore_mismatched_sizes=True,
    hidden_dropout_prob=0.3,
    attention_probs_dropout_prob=0.2
).to(device)

criterion  = nn.CrossEntropyLoss(weight=class_weights_tensor)
optimizer  = AdamW(model.parameters(), lr=LR, weight_decay=0.05)
total_steps = len(train_loader) * EPOCHS
scheduler  = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=int(total_steps * 0.1),
    num_training_steps=total_steps
)

def evaluate(model, loader):
    model.eval()
    all_preds, all_labels, total_loss = [], [], 0
    with torch.no_grad():
        for batch in loader:
            ids    = batch['input_ids'].to(device)
            mask   = batch['attention_mask'].to(device)
            labels = batch['label'].to(device)
            out    = model(input_ids=ids, attention_mask=mask)
            loss   = criterion(out.logits, labels)
            total_loss += loss.item()
            preds  = torch.argmax(out.logits, dim=1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
```

```
    f1_per = f1_score(all_labels, all_preds, average=None,
zero_division=0)
    return (total_loss / len(loader),
            accuracy_score(all_labels, all_preds),
            f1_score(all_labels, all_preds, average='macro',
zero_division=0),
            f1_per)

best_f1, patience_count, PATIENCE = 0, 0, 3
history = {'train_loss': [], 'val_loss': [], 'val_f1': []}

print('\nTRAINING START')
print(f'   Epochs  : {EPOCHS}')
print(f'   Patience: {PATIENCE}')

for epoch in range(EPOCHS):
    model.train()
    total_loss = 0
    for batch in train_loader:
        optimizer.zero_grad()
        ids    = batch['input_ids'].to(device)
        mask   = batch['attention_mask'].to(device)
        labels = batch['label'].to(device)
        out    = model(input_ids=ids, attention_mask=mask)
        loss   = criterion(out.logits, labels)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()
        total_loss += loss.item()

    train_loss = total_loss / len(train_loader)
    val_loss, val_acc, val_f1, f1_per = evaluate(model, val_loader)

    history['train_loss'].append(train_loss)
    history['val_loss'].append(val_loss)
    history['val_f1'].append(val_f1)

    print(f'\nEpoch {epoch+1}/{EPOCHS}')
    print(f'  Loss  : {train_loss:.4f} (train) → {val_loss:.4f}
(val)')
    print(f'  F1 Macro : {val_f1:.4f}  |  Acc : {val_acc:.4f}')
    print(f'  F1 per kelas: ' +
          ' | '.join(f'{label_names[i]}:{f1_per[i]:.3f}' for i in
range(len(label_names))))

    if val_f1 > best_f1:
```

```
        best_f1 = val_f1
        patience_count = 0
        model.save_pretrained('/content/indobert_best')
        tokenizer.save_pretrained('/content/indobert_best')
        print(f'  Model terbaik disimpan (F1={best_f1:.4f})')
    else:
        patience_count += 1
        print(f'  Tidak ada peningkatan
({patience_count}/{PATIENCE})')
        if patience_count >= PATIENCE:
            print('  Early stopping.')
            break

print('\nTEST SET EVALUATION')
best_model = AutoModelForSequenceClassification.from_pretrained(
    '/content/indobert_best').to(device)
best_model.eval()

all_preds, all_labels = [], []
with torch.no_grad():
    for batch in test_loader:
        out  = best_model(batch['input_ids'].to(device),
                          batch['attention_mask'].to(device))
        all_preds.extend(torch.argmax(out.logits,
dim=1).cpu().numpy())
        all_labels.extend(batch['label'].numpy())

print('\nClassification Report:')
print(classification_report(all_labels, all_preds,
target_names=label_names, zero_division=0))

fig, ax1 = plt.subplots(figsize=(8, 4))
ep = range(1, len(history['train_loss']) + 1)
ax1.plot(ep, history['train_loss'], 'b-o', label='Train Loss')
ax1.plot(ep, history['val_loss'],   'r-o', label='Val Loss')
ax1.set_xlabel('Epoch'); ax1.set_ylabel('Loss')
ax1.legend(loc='upper left')
ax2 = ax1.twinx()
ax2.plot(ep, history['val_f1'], 'g-s', label='Val F1 Macro')
ax2.set_ylabel('F1 Macro')
ax2.legend(loc='upper right')
plt.title('Riwayat Training IndoBERT — Berita Sawit')
plt.tight_layout(); plt.show()

cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu',
```

```
            xticklabels=label_names, yticklabels=label_names)
plt.title('Confusion Matrix — Test Set')
plt.ylabel('Aktual'); plt.xlabel('Prediksi')
plt.tight_layout(); plt.show()

print(f'\nBest Val F1 Macro: {best_f1:.4f}')
print('Model tersimpan di /content/indobert_best')
```