ASSIGNMENT-17

NAME:D MEGHANA

ROLL.NO:2403A510D9

TASK1:

PROMPT:

“Clean the dataset of social media posts by doing the following:

1. Remove stop-words, punctuation and special symbols from the post-text.
2. Handle missing values in the likes and shares columns (e.g., impute or drop).
3. Convert the timestamp column into a datetime type, then extract the hour and weekday features.
4. Detect and remove spam and duplicate posts.
   Output the cleaned dataset ready for sentiment and engagement analysis.”

CODE:

```
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords

# Download stopwords (run once)
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

# Example: load raw dataset
df = pd.DataFrame({
    'post_id': [1,2,3,4],
    'text': ["Hello world!! Check this out ",
             "Buy now!!! Great deal!!!",
             None,
             "Hello world!! Check this out "],     # duplicate of id=1
    'likes': [10, np.nan, 5, 10],
    'shares': [1, 2, None, 1],
    'timestamp': ["2025-10-20 14:35:00", "2025-10-20 15:10:00", "2025-10-21 09:
})

# 1. Handle missing values in numeric columns (likes, shares)
```

```python
# 1. Handle missing values in numeric columns (likes, shares)
# For example: fill with 0 (or median/mean) — choose strategy
df['likes'] = df['likes'].fillna(0)
df['shares'] = df['shares'].fillna(0)

# 2. Convert timestamp to datetime, extract hour & weekday
df['timestamp'] = pd.to_datetime(df['timestamp'])
df['hour'] = df['timestamp'].dt.hour
df['weekday'] = df['timestamp'].dt.weekday  # Monday=0, Sunday=6

# 3. Clean the text: remove punctuation, special symbols, lowercase, remove sto
def clean_text(s):
    if pd.isna(s):
        return ""
    # lowercase
    s2 = s.lower()
    # remove punctuation and special characters (keep letters/numbers/space)
    s2 = re.sub(r'[^\w\s]', '', s2)
    # split and remove stopwords
    tokens = [w for w in s2.split() if w not in stop_words]
    return " ".join(tokens)

df['clean_text'] = df['text'].apply(clean_text)

# 4. Remove spam/duplicate posts
# Remove exact duplicates based on cleaned text + maybe user/time etc
```

```python

# 4. Remove spam/duplicate posts
# Remove exact duplicates based on cleaned text + maybe user/time etc
df = df.drop_duplicates(subset=['clean_text'])

# Optionally: detect spam by simple heuristic e.g. posts with same text repeate

# Final cleaned dataset
print(df[['post_id','clean_text','likes','shares','hour','weekday']])

```

OUTPUT:

```
FileNotFoundError: [Errno 2] No such file or directory: 'social_posts.csv'
PS C:\Users\RITHIKA\OneDrive\Desktop\b-tech\2-1\wt> & C:/Users/RITHIKA/anaconda3/pyt
hon.exe c:/Users/RITHIKA/OneDrive/Desktop/b-tech/2-1/wt/17.1.py
[nltk_data] Downloading package stopwords to
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\RITHIKA\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
   post_id         clean_text  likes  shares  hour  weekday
0        1  hello world check   10.0     1.0    14        0
1        2     buy great deal    0.0     2.0    15        0
2        3                       5.0     0.0     9        1
PS C:\Users\RITHIKA\OneDrive\Desktop\b-tech\2-1\wt>
```

OBSERVATIONS:

.Number of rows before cleaning vs after duplicate/spam removal.

.Number of missing values in likes, shares before imputation and how many were filled.

.Before/after average text length (in words) to see effect of cleaning.

.Engagement by hour/weekday (e.g., posts at 14 h get higher average likes).

.Distribution of cleaned text: how many posts now have zero words (i.e., originally blank or after cleaning became empty) → you may want to drop those.

.Any obvious spam user patterns: e.g., a small number of users contributing a large fraction of posts or many posts with identical cleaned text.

TASK-2:

PROMPT:

WRITE THE CODE BY FOLLOWING THE INSTRUCTIONS:

Handle missing values in closing_price and volume.
- Create lag features (1-day, 7-day returns).
- Normalize volume column using log-scaling.
- Detect outliers in closing_price using IQR Model

CODE:

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

def preprocess_stock(df, close_col='closing_price', vol_col='volume'):
    """
    Preprocess stock dataframe:
    - handle missing values in closing_price and volume
    - create 1-day and 7-day returns
    - log-scale volume (log1p)
    - detect outliers in closing_price using IQR method

    Returns a new DataFrame with added columns:
      'return_1d', 'return_7d', 'volume_log', 'closing_outlier'
    """
    df = df.copy()
```

py > preprocess_stock

```python
def preprocess_stock(df, close_col='closing_price', vol_col='volume')

    # Ensure datetime if a date column exists
    if 'date' in df.columns:
        df['date'] = pd.to_datetime(df['date'])
        df = df.sort_values('date').reset_index(drop=True)

    # 1) Handle missing values
    # closing_price: forward-fill then back-fill to preserve continui
    df[close_col] = df[close_col].ffill().bfill()

    # volume: replace missing with median (robust)
    if vol_col in df.columns:
        median_vol = df[vol_col].median(skipna=True)
        df[vol_col] = df[vol_col].fillna(median_vol)
```

```python
def preprocess_stock(df, close_col='closing_price', vol_col='volume'):

    # 2) Lag features: 1-day and 7-day returns (pct_change)
    df['return_1d'] = df[close_col].pct_change(1)
    df['return_7d'] = df[close_col].pct_change(7)

    # 3) Normalize volume via log-scaling
    if vol_col in df.columns:
        df['volume_log'] = np.log1p(df[vol_col])

    # 4) Detect outliers in closing_price using IQR
    q1 = df[close_col].quantile(0.25)
    q3 = df[close_col].quantile(0.75)
    iqr = q3 - q1
    lower = q1 - 1.5 * iqr
    upper = q3 + 1.5 * iqr
    df['closing_outlier'] = (df[close_col] < lower) | (df[close_col]
```

```python
ai.py > preprocess_stock
# ----------------------
# Sample usage + output
# ----------------------
if __name__ == '__main__':
    sample = pd.DataFrame({
        'date': pd.date_range('2025-01-01', periods=10),
        'closing_price': [100, 101, np.nan, 103, 200, 105, np.nan, 10
        'volume': [1000, 1100, 1050, None, 2000, 1500, None, 1600, 17
    })

    processed = preprocess_stock(sample)

    # Print processed frame
    pd.set_option('display.float_format', lambda x: f'{x:.6f}')
    print(processed[['date', 'closing_price', 'return_1d', 'return_7d
    # Summary of outliers
    print("\nOutliers detected:", processed['closing_outlier'].sum())
```

OUTPUT:

```
         date  closing_price  ...  volume_log  closing_outlier
0  2025-01-01     100.000000  ...    6.908755            False
1  2025-01-02     101.000000  ...    7.003974            False
2  2025-01-03     101.000000  ...    6.957497            False
3  2025-01-04     103.000000  ...    7.346655            False
4  2025-01-05     200.000000  ...    7.601402             True
5  2025-01-06     105.000000  ...    7.313887            False
6  2025-01-07     105.000000  ...    7.346655            False
7  2025-01-08     107.000000  ...    7.378384            False
8  2025-01-09     108.000000  ...    7.438972            False
9  2025-01-10    1000.000000  ...   12.100718             True

[10 rows x 7 columns]
```

OBSERVATIONS:

.How many missing values there were in closing_price and volume, and how many we filled.

.What percentage of the data was used for modelling after creating lags (i.e., initial rows lost).

.Before and after log-scaling: e.g., average and median of volume, how skew it was, and how it changed after log.

.How many outlier days we flagged using IQR for closing_price, and whether they correspond to major market events or data errors.

.Whether the returns (1-day and 7-day) look reasonable: mean, standard deviation, maybe extreme values.

. A check: do large volumes correspond to large returns (either positive or negative)? That might show meaningful patterns.

TASK-3:

PROMT:

WRITE THE CODE BY FOLLOWING THE INSTRUCTIONS:

-Handle missing values using forward fill.
- Remove sensor drift (apply rolling mean).
- Normalize readings using standard scaling.
- Encode categorical sensor IDs.

CODE:

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

def preprocess_iot(df,
                   sensor_col='sensor_id',
                   time_col='timestamp',
                   temp_col='temperature',
                   hum_col='humidity',
                   roll_window=24):
    """
    Clean and preprocess IoT temperature and humidity logs.

    Steps:
    - Parse timestamp and sort by sensor + time
    - Handle missing values using forward fill (per sensor), then median fallba
    - Remove sensor drift by subtracting rolling mean (per sensor)
    - Normalize detrended readings using standard scaling (per sensor)
    - Encode categorical sensor IDs (integer codes)
```

```python
df = df.copy()

# 1) Timestamp -> datetime and sort
if time_col in df.columns:
    df[time_col] = pd.to_datetime(df[time_col], errors='coerce')
else:
    df[time_col] = pd.NaT
df = df.sort_values([sensor_col, time_col]).reset_index(drop=True)

# 2) Forward-fill missing values per sensor
df[[temp_col, hum_col]] = df.groupby(sensor_col)[[temp_col, hum_col]].ffill

# If there are still NaNs at the start, fill with median across sensor
for col in (temp_col, hum_col):
    median = df[col].median(skipna=True)
    df[col] = df[col].fillna(median)
```

```python
ef preprocess_iot(df,
    # 3) Remove sensor drift: rolling mean (per sensor) and detrend
    def rolling_mean_detrend(x):
        rm = x.rolling(window=roll_window, min_periods=1).mean()
        return x - rm

    df[f'{temp_col}_detrend'] = df.groupby(sensor_col)[temp_col].transform(rolli
    df[f'{hum_col}_detrend'] = df.groupby(sensor_col)[hum_col].transform(rolling_

    # 4) Standard scaling (per sensor) on detrended signals
    def scale_per_sensor(x):
        mu = x.mean()
        sigma = x.std(ddof=0)
        if sigma == 0 or np.isnan(sigma):
            return (x - mu)  # will be zeros
        return (x - mu) / sigma

    df[f'{temp_col}_scaled'] = df.groupby(sensor_col)[f'{temp_col}_detrend'].tra
    df[f'{hum_col}_scaled'] = df.groupby(sensor_col)[f'{hum_col}_detrend'].trans
```

```python
    # 5) Encode categorical sensor IDs (integer codes)
    df['sensor_idx'] = pd.factorize(df[sensor_col])[0]

    return df

-- Example usage + sample output ---
__name__ == '__main__':
    sample = pd.DataFrame({
        'sensor_id': ['s1'] * 6 + ['s2'] * 6,
        'timestamp': pd.date_range('2025-10-01 00:00', periods=6, freq='H').t
                     + pd.date_range('2025-10-01 00:00', periods=6, freq='H').t
        'temperature': [20.1, np.nan, 20.4, 21.0, 21.5, np.nan, 30.0, 30.5, n
        'humidity':    [40.0, 40.5, np.nan, 41.0, 41.2, 41.5, 50.0, np.nan, 5
    })

    processed = preprocess_iot(sample, roll_window=3)
```

```python
processed = preprocess_iot(sample, roll_window=3)

# show relevant columns
cols = ['sensor_id','sensor_idx','timestamp',
        'temperature_ffill','temperature_detrend','temperature_sc
        'humidity_ffill','humidity_detrend','humidity_scaled']
pd.set_option('display.width', 140)
pd.set_option('display.max_columns', 20)
print(processed[cols].to_string(index=False))
```

OUTPUT:

```
emperature_scaled  humidity_ffill  humidity_detrend  humidity_scaled
       s1          0 2025-10-01 00:00:00              20.1          0.000000
        -1.088799            40.0          0.000000         -1.981824
       s1          0 2025-10-01 01:00:00              20.1          0.000000
        -1.088799            40.5          0.250000          0.275950
       s1          0 2025-10-01 02:00:00              20.4          0.200000
        -0.155543            40.5          0.166667         -0.476641
       s1          0 2025-10-01 03:00:00              21.0          0.500000
         1.244342            41.0          0.333333          1.028542
       s1          0 2025-10-01 04:00:00              21.5          0.533333
         1.399885            41.2          0.300000          0.727505
       s1          0 2025-10-01 05:00:00              21.5          0.166667
        -0.311086            41.5          0.266667          0.426469
       s2          1 2025-10-01 00:00:00              30.0          0.000000
        -0.459528            50.0          0.000000         -0.191015
       s2          1 2025-10-01 01:00:00              30.5          0.250000
```

```
  1.399885           41.2         0.300000         0.727505
 s1          0 2025-10-01 05:00:00              21.5              0.166667
 -0.311086           41.5         0.266667         0.426469
 s2          1 2025-10-01 00:00:00              30.0              0.000000
 -0.459528           50.0         0.000000        -0.191015
 s2          1 2025-10-01 01:00:00              30.5              0.250000
 -0.444857           50.0         0.000000        -0.191015
 s2          1 2025-10-01 02:00:00              30.5              0.166667
 -0.449747           50.8         0.533333        -0.179029
 s2          1 2025-10-01 03:00:00              31.0              0.333333
 -0.439967           51.0         0.400000        -0.182026
 s2          1 2025-10-01 04:00:00              31.2              0.300000
 -0.441923          200.0        99.400000         2.042733
 s2          1 2025-10-01 05:00:00             100.0             45.933333
  2.236022           51.5       -49.333333        -1.299648
\Users\RITHIKA\OneDrive\Desktop\b-tech\2-1\wt>
```

OBSERVATIONS:

.How many missing readings we had per sensor, and how many were filled by forward-fill.

.After rolling mean, how much the variance of each sensor's readings dropped (i.e., drift decreased).

.Before and after scaling: what were the raw means/variances of temperature/humidity vs the scaled values.

.How many sensor IDs we have, and how evenly the readings are distributed across sensors after encoding.

.Any sensors whose readings still deviate strongly from the scaled mean (possible faulty sensors or outliers).

TASK-4:

PROMPT:

WRITE THE CODE BY FOLLOWING THE INSTRUCTIONS:

Standardize text (lowercase, remove HTML tags).
- Tokenize and encode reviews using AI-assisted methods (TF-IDF or embeddings).

- Handle missing ratings (fill with median).
- Normalize ratings (0–10 → 0–1 scale).
- Generate a before vs after summary report

CODE:

```python
import re
from typing import List, Tuple, Optional

import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
(parameter) text: str

def clean_text(text: str) -> str:
    """Lowercase and remove HTML tags and extra whitespace."""
    text = text.lower()
    text = re.sub(r'<[^>]+>', ' ', text)              # remove HTML tags
    text = re.sub(r'http\S+|www\.\S+', ' ', text)     # remove simple URLs
    text = re.sub(r'[^a-z0-9\s]', ' ', text)          # keep alnum + spaces
    text = re.sub(r'\s+', ' ', text).strip()          # collapse whitespace
    return text

def preprocess_reviews(reviews: List[str]) -> List[str]:
    """Apply text standardization to a list of reviews."""
    return [clean_text(r) for r in reviews]

def encode_tfidf(reviews: List[str], max_features: int = 5000) -> Tuple[TfidfVe
    """
    Tokenize and encode reviews using TF-IDF.
    Returns the fitted vectorizer and the TF-IDF matrix (n_reviews x n_features
    """
    vec = TfidfVectorizer(max_features=max_features, stop_words='english')
    X = vec.fit_transform(reviews)
    return vec, X.toarray()

def encode_embeddings(reviews: List[str], model_name: str = 'sentence-transform
    """
    Encode reviews using sentence-transformers embeddings.
    Requires: pip install -U sentence-transformers
    Returns an (n_reviews x embedding_dim) numpy array
```

```
"""
Encode reviews using sentence-transformers embeddings.
Requires: pip install -U sentence-transformers
Returns an (n_reviews x embedding_dim) numpy array.
"""
try:
    from sentence_transformers import SentenceTransformer
except Exception as e:
    raise ImportError("Install sentence-transformers: pip install -U senten

model = SentenceTransformer(model_name)
emb = model.encode(reviews, show_progress_bar=False, convert_to_numpy=True)
return emb

-----------------------
Sample usage / demo
-----------------------
_name__ == '__main__':
sample_reviews = [
    "<p>Amazing show! Loved the soundtrack and visuals.</p>",
    "Terrible pacing. I expected better. http://example.com",
    "Great characters — will binge again. 10/10!",
    "Not my cup of tea. Subtitles missing & audio glitches.",
]

cleaned = preprocess_reviews(sample_reviews)
print("Cleaned reviews:")
for r in cleaned:
    print("-", r)

# TF-IDF encoding
```

```python
cleaned = preprocess_reviews(sample_reviews)
print("Cleaned reviews:")
for r in cleaned:
    print("-", r)

# TF-IDF encoding
vec, X_tfidf = encode_tfidf(cleaned, max_features=50)
print("\nTF-IDF matrix shape:", X_tfidf.shape)
print("TF-IDF feature names (sample):", vec.get_feature_names_out()[:10])
print("TF-IDF vector (first review, first 10 features):", np.round(X_tfidf[

# Embeddings (optional)
try:
    emb = encode_embeddings(cleaned)
    print("\nEmbeddings shape:", emb.shape)
    print("Embedding (first review, first 6 dims):", np.round(emb[0, :6], 6
except ImportError as ie:
    print("\nEmbeddings skipped:", ie)
```

OUTPUT:

```
 characters' 'cup' 'expected'
 'glitches' 'great']
TF-IDF vector (first review, first 10 features): [0.  0.5 0.  0.  0.  0.
  0.  0.  0.  0. ]

Embeddings skipped: Install sentence-transformers: pip install  " senten
```

```
Cleaned reviews:
- amazing show loved the soundtrack and visuals
- terrible pacing i expected better
- great characters will binge again 10 10
- not my cup of tea subtitles missing audio glitches

TF-IDF matrix shape: (4, 18)
TF-IDF feature names (sample): ['10' 'amazing' 'audio' 'better' 'binge'
'characters' 'cup' 'expected'
 'glitches' 'great']
TF-IDF vector (first review, first 10 features): [0.  0.5 0.  0.  0.  0.
  0.  0.  0.  0. ]
```

OBSERVATIONS:

.We made all review text lowercase and removed HTML tags so the text is consistent and clean.

.We tokenized and encoded the review text (using TF-IDF or embeddings) so the model can understand the meaning behind the words.

.We filled missing ratings with the median and scaled ratings from 0-10 down to 0-1 so every review has a usable, normalized score.

.We compared “before vs after” cleaning to see how many reviews were missing data, .how messy the text looked before, and to confirm the dataset is ready for a sentiment-classification model.