ASSIGNMENT-17.2

NAME:G HARSHA VARDHAN REDDY

ROLL NO :2403A52313

COURSE:AI ASSISTED CODING

BATCH:01

Task 1 – Social Media Data Cleaning

Task: Clean raw social media posts dataset.
Instructions:
- Remove stopwords, punctuation, and special symbols from post text.
- Handle missing values in likes and shares columns.
- Convert timestamp to datetime and extract features (hour, weekday).
- Detect and remove spam/duplicate posts.

Expected Output: A cleaned dataset with structured features for

sentiment/engagement analysis

Prompt:

Generate Social Media Data Cleaning
Task: Clean raw social media posts
dataset.
Instructions:
- Remove stopwords, punctuation, and special symbols from post text.
- Handle missing values in likes and shares columns.
- Convert timestamp to datetime and extract features (hour, weekday). - Detect and remove spam/duplicate posts

Code:

```
import pandas as pd
import numpy as np

# Assuming the dataset is loaded into a DataFrame 'df' with columns: 'date', 'closing_price', 'volume'
# Example: df = pd.read_csv('stock_data.csv')
# For demonstration, I'll create a sample dataset. In practice, replace this with your actual data loac

# Sample data creation (replace with actual data loading)
dates = pd.date_range('2020-01-01', periods=100, freq='D')
np.random.seed(42)
closing_prices = np.random.normal(100, 10, 100)  # Simulated closing prices
volumes = np.random.poisson(1000000, 100)  # Simulated volumes
df = pd.DataFrame({'date': dates, 'closing_price': closing_prices, 'volume': volumes})

# Introduce some missing values for demonstration
df.loc[10:15, 'closing_price'] = np.nan
df.loc[20:25, 'volume'] = np.nan

# Set date as index for time-series handling
df.set_index('date', inplace=True)
df.sort_index(inplace=True)

# Step 1: Handle missing values
# For closing_price: Use forward fill (assumes prices are continuous)
df['closing_price'].fillna(method='ffill', inplace=True)

# For volume: Use median imputation (robust to outliers)
df['volume'].fillna(df['volume'].median(), inplace=True)
```

bles Terminal

```python
# The preprocessed dataset is now ready for forecasting models
# It includes original columns, lag features, normalized volume, and outlier flags
print(df.head(10))  # Display first 10 rows for verification
```

```
...         closing_price     volume  1_day_return  7_day_return  \
date
2020-01-01     104.967142   999766.0           NaN           NaN
2020-01-02      98.617357   998640.0     -0.060493           NaN
2020-01-03     106.476885  1002095.0      0.079697           NaN
2020-01-04     115.230299  1000052.0      0.082210           NaN
2020-01-05      97.658466   999608.0     -0.152493           NaN
2020-01-06      97.658630  1002717.0      0.000002           NaN
2020-01-07     115.792128   999992.0      0.185682           NaN
2020-01-08     107.674347   999361.0     -0.070107      0.025791
2020-01-09      95.305256  1000312.0     -0.114875     -0.033585
2020-01-10     105.425600   997771.0      0.106189     -0.009873

            volume_normalized  is_outlier
date
2020-01-01          13.815278       False
2020-01-02          13.814151       False
2020-01-03          13.817604       False
2020-01-04          13.815564       False
2020-01-05          13.815119       False
2020-01-06          13.818225       False
2020-01-07          13.815504       False
2020-01-08          13.814872       False
2020-01-09          13.815824       False
2020-01-10          13.813280       False
/tmp/ipython-input-1102876934.py:25: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inpl
```

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Assuming the dataset is loaded into a DataFrame 'df' with columns: 'timestamp', 'sensor_id', 'temperature', 'humidity'
# Example: df = pd.read_csv('iot_sensor_logs.csv')
# For demonstration, I'll create a sample dataset. In practice, replace this with your actual data loading.

# Sample data creation (replace with actual data loading)
np.random.seed(42)
timestamps = pd.date_range('2023-01-01', periods=100, freq='H')
sensor_ids = np.random.choice(['Sensor_A', 'Sensor_B', 'Sensor_C'], 100)
temperatures = np.random.normal(25, 5, 100) + np.sin(np.arange(100) * 0.1)  # Simulated with some drift
humidities = np.random.normal(60, 10, 100) + np.cos(np.arange(100) * 0.1)
df = pd.DataFrame({'timestamp': timestamps, 'sensor_id': sensor_ids, 'temperature': temperatures, 'humidity': humidities})

# Introduce some missing values for demonstration
df.loc[10:15, 'temperature'] = np.nan
df.loc[20:25, 'humidity'] = np.nan

# Set timestamp as index for time-series handling
df.set_index('timestamp', inplace=True)
df.sort_index(inplace=True)

# Step 1: Handle missing values using forward fill
df['temperature'].fillna(method='ffill', inplace=True)
df['humidity'].fillna(method='ffill', inplace=True)
```

es Terminal

```
# The structured dataset is now optimized for anomaly detection
# It includes smoothed and normalized readings, encoded sensor IDs, and is ready for models like Isolation Forest or Autoencoders
print(df.head(10))  # Display first 10 rows for verification
```

```
                    sensor_id  temperature   humidity  temperature_smoothed  \
timestamp
2023-01-01 00:00:00  Sensor_C    27.910614  57.755090             27.910614
2023-01-01 01:00:00  Sensor_A    29.538576  58.936337             29.538576
2023-01-01 02:00:00  Sensor_C    29.670331  46.579652             29.670331
2023-01-01 03:00:00  Sensor_C    29.070509  72.862609             29.070509
2023-01-01 04:00:00  Sensor_A    24.353589  73.915029             24.353589
2023-01-01 05:00:00  Sensor_A    22.362039  52.206121             26.063209
2023-01-01 06:00:00  Sensor_C    18.023876  67.001744             25.645289
2023-01-01 07:00:00  Sensor_B    31.142453  72.935550             25.064572
2023-01-01 08:00:00  Sensor_C    24.828695  62.959589             24.470680
2023-01-01 09:00:00  Sensor_C    23.731410  69.095624             23.936770

                     humidity_smoothed  temperature_normalized  \
timestamp
2023-01-01 00:00:00          57.755090                0.928742
2023-01-01 01:00:00          58.936337                1.586458
2023-01-01 02:00:00          46.579652                1.639689
2023-01-01 03:00:00          72.862609                1.397354
2023-01-01 04:00:00          73.915029               -0.508340
2023-01-01 05:00:00          63.424735                0.182368
2023-01-01 06:00:00          63.878089                0.013523
2023-01-01 07:00:00          62.812960               -0.221094
2023-01-01 08:00:00          65.240578               -0.461033
2023-01-01 09:00:00          65.306172               -0.676739
```

```
print("Cleaned DataFrame:")
print(df.head())
print("\nSummary Report:")
print(report)
```

```
Cleaned DataFrame:
           review_text  rating cleaned_review  rating_normalized  \
0  <b>Great movie!</b>     8.0   great movie!                0.8
1      Bad <i>film</i>     8.0       bad film                0.8
2                  NaN     6.0            nan                0.6
3                 Okay     9.0           okay                0.9

                                        tfidf_vector
0  [0.0, 0.0, 0.7071067811865476, 0.7071067811865...
1  [0.7071067811865476, 0.7071067811865476, 0.0, ...
2                     [0.0, 0.0, 0.0, 0.0, 1.0, 0.0]
3                     [0.0, 0.0, 0.0, 0.0, 0.0, 1.0]

Summary Report:
{'before_avg_words': np.float64(1.5), 'after_avg_words': np.float64(1.5), 'before_rating_mean': np.float64(7.75), 'after_rating_mean': np.float64(0.775), 'rating_median': 8.0
/tmp/ipython-input-522370084.py:19: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the o


  df['cleaned_review'].fillna('', inplace=True)
/tmp/ipython-input-522370084.py:24: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.
```

Task 2 – Financial Data Preprocessing

Task: Preprocess a stock market dataset.
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 method.

Expected Output: A time-series dataset ready for forecasting models

Prompt:

Generate a code for Financial Data Preprocessing
Task: Preprocess a stock market dataset. 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 method

Code:

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

# Assuming the dataset is loaded into a DataFrame 'df' with columns: 'date', 'closing_price', 'volume'
# Example: df = pd.read_csv('stock_data.csv')
# For demonstration, I'll create a sample dataset. In practice, replace this with your actual data loading.

# Sample data creation (replace with actual data loading)
dates = pd.date_range('2020-01-01', periods=100, freq='D')
np.random.seed(42)
closing_prices = np.random.normal(100, 10, 100)  # Simulated closing prices
volumes = np.random.poisson(1000000, 100)  # Simulated volumes
df = pd.DataFrame({'date': dates, 'closing_price': closing_prices, 'volume': volumes})

# Introduce some missing values for demonstration
df.loc[10:15, 'closing_price'] = np.nan
df.loc[20:25, 'volume'] = np.nan

# Set date as index for time-series handling
df.set_index('date', inplace=True)
df.sort_index(inplace=True)

# Step 1: Handle missing values
# For closing_price: Use forward fill (assumes prices are continuous)
df['closing_price'].fillna(method='ffill', inplace=True)

# For volume: Use median imputation (robust to outliers)
df['volume'].fillna(df['volume'].median(), inplace=True)
```

```
            closing_price     volume  1_day_return  7_day_return  \
date
2020-01-01     104.967142   999766.0           NaN           NaN
2020-01-02      98.617357   998640.0     -0.060493           NaN
2020-01-03     106.476885  1002095.0      0.079697           NaN
2020-01-04     115.230299  1000052.0      0.082210           NaN
2020-01-05      97.658466   999608.0     -0.152493           NaN
2020-01-06      97.658630  1002717.0      0.000002           NaN
2020-01-07     115.792128   999992.0      0.185682           NaN
2020-01-08     107.674347   999361.0     -0.070107      0.025791
2020-01-09      95.305256  1000312.0     -0.114875     -0.033585
2020-01-10     105.425600   997771.0      0.106189     -0.009873

            volume_normalized  is_outlier
date
2020-01-01          13.815278       False
2020-01-02          13.814151       False
2020-01-03          13.817604       False
2020-01-04          13.815564       False
2020-01-05          13.815119       False
2020-01-06          13.818225       False
2020-01-07          13.815504       False
2020-01-08          13.814872       False
2020-01-09          13.815824       False
2020-01-10          13.813280       False
/tmp/ipython-input-1102876934.py:25: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the op
```

Terminal 14:13 Python 3

Task 3 – IoT Sensor Data Preparation

Task: Clean and preprocess IoT temperature and humidity logs.
Instructions:
- Handle missing values using forward fill.
- Remove sensor drift (apply rolling mean).
- Normalize readings using standard scaling. - Encode categorical sensor IDs.

Expected Output: A structured dataset optimized for anomaly

detection

Prompt:

Generate a python code for IoT Sensor Data Preparation Task:
Clean and preprocess IoT temperature and humidity logs.
Instructions:
- Handle missing values using forward fill.
- Remove sensor drift (apply rolling mean).
- Normalize readings using standard scaling. - Encode categorical sensor ID

Code:

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Assuming the dataset is loaded into a DataFrame 'df' with columns: 'timestamp', 'sensor_id', 'temperature', 'humidity'
# Example: df = pd.read_csv('iot_sensor_logs.csv')
# For demonstration, I'll create a sample dataset. In practice, replace this with your actual data loading.

# Sample data creation (replace with actual data loading)
np.random.seed(42)
timestamps = pd.date_range('2023-01-01', periods=100, freq='H')
sensor_ids = np.random.choice(['Sensor_A', 'Sensor_B', 'Sensor_C'], 100)
temperatures = np.random.normal(25, 5, 100) + np.sin(np.arange(100) * 0.1)  # Simulated with some drift
humidities = np.random.normal(60, 10, 100) + np.cos(np.arange(100) * 0.1)
df = pd.DataFrame({'timestamp': timestamps, 'sensor_id': sensor_ids, 'temperature': temperatures, 'humidity': humidities})

# Introduce some missing values for demonstration
df.loc[10:15, 'temperature'] = np.nan
df.loc[20:25, 'humidity'] = np.nan

# Set timestamp as index for time-series handling
df.set_index('timestamp', inplace=True)
df.sort_index(inplace=True)

# Step 1: Handle missing values using forward fill
df['temperature'].fillna(method='ffill', inplace=True)
df['humidity'].fillna(method='ffill', inplace=True)
```

Terminal 14:13 Python 3

```python
# The structured dataset is now optimized for anomaly detection
# It includes smoothed and normalized readings, encoded sensor IDs, and is ready for models like Isolation Forest or Autoencoders
print(df.head(10))  # Display first 10 rows for verification
```

```
timestamp
2023-01-01 00:00:00  Sensor_C    27.910614  57.755090           27.910614
2023-01-01 01:00:00  Sensor_A    29.538576  58.936337           29.538576
2023-01-01 02:00:00  Sensor_C    29.670331  46.579652           29.670331
2023-01-01 03:00:00  Sensor_C    29.070509  72.862609           29.070509
2023-01-01 04:00:00  Sensor_A    24.353589  73.915029           24.353589
2023-01-01 05:00:00  Sensor_A    22.362039  52.206121           26.063209
2023-01-01 06:00:00  Sensor_C    18.023876  67.001744           25.645289
2023-01-01 07:00:00  Sensor_B    31.142453  72.935550           25.064572
2023-01-01 08:00:00  Sensor_C    24.828695  62.959589           24.470680
2023-01-01 09:00:00  Sensor_C    23.731410  69.095624           23.936770

                     humidity_smoothed  temperature_normalized  \
timestamp
2023-01-01 00:00:00          57.755090                0.928742
2023-01-01 01:00:00          58.936337                1.586458
2023-01-01 02:00:00          46.579652                1.639689
2023-01-01 03:00:00          72.862609                1.397354
2023-01-01 04:00:00          73.915029               -0.508340
2023-01-01 05:00:00          63.424735                0.182368
2023-01-01 06:00:00          63.878089                0.013523
2023-01-01 07:00:00          62.812960               -0.221094
2023-01-01 08:00:00          65.240578               -0.461033
2023-01-01 09:00:00          65.306172               -0.676739

                     humidity_normalized  sensor_id_encoded
```

Terminal

Task 4 – Real-Time Application: Movie Reviews Data Cleaning
Task: A streaming platform wants to analyze customer reviews.
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.

Expected Output: A cleaned dataset ready for sentiment classification.

Deliverables (For All Tasks)

1. AI-generated prompts for code and test case generation.
2. At least 3 assert test cases for each task.
3. AI-generated initial code and execution screenshots.
4. Analysis of whether code passes all tests.
5. Improved final version with inline comments and explanations.
6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.

Prompt:

Generate A streaming platform wants to analyze customer reviews. 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 pandas as pd
import numpy as np
import re
from sklearn.feature_extraction.text import TfidfVectorizer

# Assuming the dataset is loaded into a DataFrame 'df' with columns: 'review_text', 'rating'
# Example: df = pd.read_csv('movie_reviews.csv')
# For demonstration, sample data is created. Replace with actual loading.

# Sample data creation (replace with actual data loading)
reviews = ['<b>Great movie!</b>', 'Bad <i>film</i>', np.nan, 'Okay']
ratings = [8.0, np.nan, 6.0, 9.0]
df = pd.DataFrame({'review_text': reviews, 'rating': ratings})

# Step 1: Standardize text (lowercase, remove HTML tags)
# Convert to lowercase and use regex to strip HTML tags for cleaner text
df['cleaned_review'] = df['review_text'].astype(str).str.lower().str.replace(r'<[^>]+>', '', regex=True).str.strip()
# Handle any remaining NaN by filling with empty string (avoids TF-IDF errors)
df['cleaned_review'].fillna('', inplace=True)

# Step 2: Handle missing ratings (fill with median)
# Calculate median and fill missing values to maintain distribution
median_rating = df['rating'].median()
df['rating'].fillna(median_rating, inplace=True)

# Step 3: Normalize ratings (0-10 → 0-1 scale)
# Simple min-max normalization assuming original range is 0-10
df['rating_normalized'] = df['rating'] / 10.0
```

```python
    'after_avg_words': df['cleaned_review'].str.split().str.len().mean(),
    'before_rating_mean': df['rating'].mean(),  # Original (with fills)
    'after_rating_mean': df['rating_normalized'].mean(),
    'rating_median': median_rating,
    'total_reviews': len(df)
}

# The cleaned dataset is now ready for sentiment classification (e.g., feed into classifiers like LogisticRegression)
print("Cleaned DataFrame:")
print(df.head())
print("\nSummary Report:")
print(report)
```

```
Cleaned DataFrame:
          review_text  rating cleaned_review  rating_normalized  \
0  <b>Great movie!</b>     8.0   great movie!                0.8
1      Bad <i>film</i>     8.0       bad film                0.8
2                  NaN     6.0            nan                0.6
3                 Okay     9.0           okay                0.9

                                        tfidf_vector
0  [0.0, 0.0, 0.7071067811865476, 0.7071067811865...
1  [0.7071067811865476, 0.7071067811865476, 0.0, ...
2                     [0.0, 0.0, 0.0, 0.0, 1.0, 0.0]
3                     [0.0, 0.0, 0.0, 0.0, 0.0, 1.0]

Summary Report:
{'before_avg_words': np.float64(1.5), 'after_avg_words': np.float64(1.5), 'before_rating_mean': np.float64(7.75), 'after_rating_mean': np.float64(0.775), 'rating_median': 8.0
/tmp/ipython-input-522370084.py:19: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.
```