LAB ASSIGNMENT – 17

NAME : M.SHIVA

HALL.NO : 2403A52377

BATCH.NO : AI 14

PROMPT 01 :

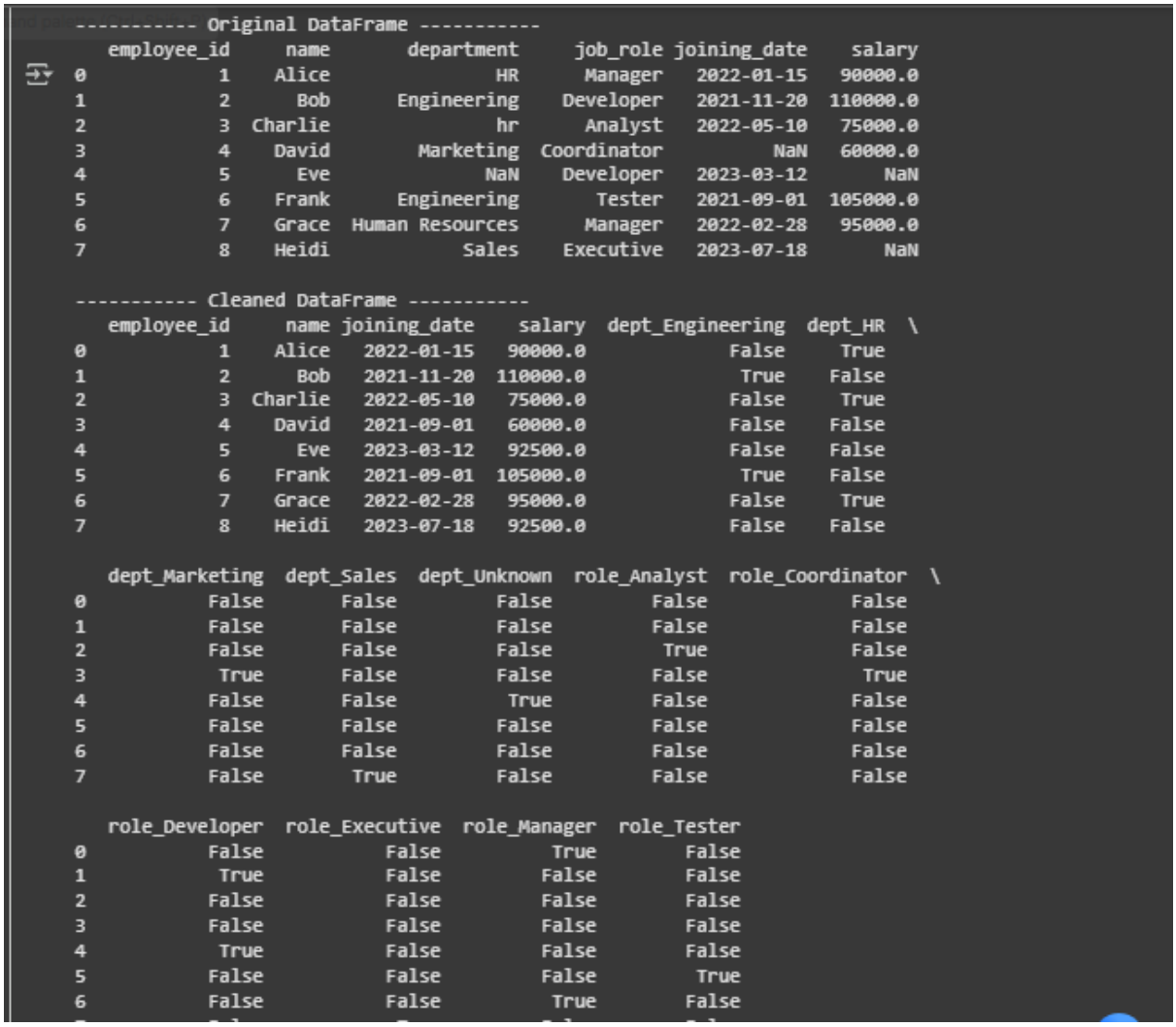
Write a Python script to clean an employee dataset. Handle missing data, standardize department names and dates, and encode the categorical columns.

CODE:

```
import pandas as pd
import numpy as np

# 1. Create a sample messy DataFrame
data = {
    'employee_id': [1, 2, 3, 4, 5, 6, 7, 8],
    'name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank',
'Grace', 'Heidi'],
    'department': ['HR', 'Engineering', 'hr', 'Marketing', np.nan,
'Engineering', 'Human Resources', 'Sales'],
    'job_role': ['Manager', 'Developer', 'Analyst', 'Coordinator',
'Developer', 'Tester', 'Manager', 'Executive'],
    'joining_date': ['2022-01-15', '2021-11-20', '2022-05-10', np.nan,
'2023-03-12', '2021-09-01', '2022-02-28', '2023-07-18'],
    'salary': [90000, 110000, 75000, 60000, np.nan, 105000, 95000,
np.nan]
}
df = pd.DataFrame(data)

print("----------- Original DataFrame -----------")
print(df)

# 2. Handle missing salary values with the median
median_salary = df['salary'].median()
df['salary'].fillna(median_salary, inplace=True)
```

```
# 3. Handle missing department values
df['department'].fillna('Unknown', inplace=True)

# 4. Convert 'joining_date' to datetime format and fill missing
df['joining_date'] = pd.to_datetime(df['joining_date'])
# Fill any remaining NaT (Not a Time) values, for instance, with the
mode
if df['joining_date'].isnull().any():
    mode_date = df['joining_date'].mode()[0]
    df['joining_date'].fillna(mode_date, inplace=True)

# 5. Standardize department names
department_mapping = {
    'hr': 'HR',
    'Human Resources': 'HR',
}
df['department'] = df['department'].replace(department_mapping)


# 6. Encode categorical variables
df_cleaned = pd.get_dummies(df, columns=['department', 'job_role'],
prefix=['dept', 'role'])

print("\n----------- Cleaned DataFrame -----------")
print(df_cleaned)
```

OUTPUT :



```
---------- Original DataFrame ----------
   employee_id     name       department     job_role joining_date    salary
0            1    Alice               HR      Manager   2022-01-15   90000.0
1            2      Bob      Engineering    Developer   2021-11-20  110000.0
2            3  Charlie               hr      Analyst   2022-05-10   75000.0
3            4    David        Marketing  Coordinator          NaN   60000.0
4            5      Eve              NaN    Developer   2023-03-12       NaN
5            6    Frank      Engineering       Tester   2021-09-01  105000.0
6            7    Grace  Human Resources      Manager   2022-02-28   95000.0
7            8    Heidi            Sales    Executive   2023-07-18       NaN

---------- Cleaned DataFrame ----------
   employee_id     name joining_date    salary  dept_Engineering  dept_HR  \
0            1    Alice   2022-01-15   90000.0             False     True
1            2      Bob   2021-11-20  110000.0              True    False
2            3  Charlie   2022-05-10   75000.0             False     True
3            4    David   2021-09-01   60000.0             False    False
4            5      Eve   2023-03-12   92500.0             False    False
5            6    Frank   2021-09-01  105000.0              True    False
6            7    Grace   2022-02-28   95000.0             False     True
7            8    Heidi   2023-07-18   92500.0             False    False

   dept_Marketing  dept_Sales  dept_Unknown  role_Analyst  role_Coordinator  \
0           False       False         False         False             False
1           False       False         False         False             False
2           False       False         False          True             False
3            True       False         False         False              True
4           False       False          True         False             False
5           False       False         False         False             False
6           False       False         False         False             False
7           False        True         False         False             False

   role_Developer  role_Executive  role_Manager  role_Tester
0           False           False          True        False
1            True           False         False        False
2           False           False         False        False
3           False           False         False        False
4            True           False         False        False
5           False           False         False         True
6           False           False          True        False
```

CODE EXPLANATION :

- Create Sample Data: A sample DataFrame is created with messy data to work with, including missing values (np.nan) and inconsistent text like "hr" and "Human Resources".
- Handle Missing Salary: Any missing salary is filled with the column's median value. The median is chosen because it is not easily skewed by unusually high or low salaries.
- Handle Missing Department: Missing department entries are filled with the placeholder 'Unknown' to ensure every row has a value.
- Convert Joining Date: The pd.to_datetime() function converts the date strings into a standard datetime format. Any missing dates are filled with the most common date in the column (the mode).

- Standardize Departments: A dictionary is used to map variations like "hr" and "Human Resources" to the standard name "HR" for consistency.
- Encode Categorical Variables: pd.get_dummies() creates new columns for each unique category in department and job_role. This process, called one-hot encoding, converts text data into a numerical format that machine learning models can understand.

PROMPT 02 :

Can you write me a Python script to clean up some sales data? It needs to fix the transaction dates, add a new column for the month and year, remove any sales that were zero or negative, and finally, scale all the transaction amounts so they fit between 0 and 1.

CODE :

```
import pandas as pd
import numpy as np

# Create a sample sales transaction DataFrame
data = {
    'transaction_id': [101, 102, 103, 104, 105, 106, 107],
    'transaction_date': ['2023-05-21', '2023-05-22', '2023-06-01',
'2023-06-05', '2023-06-10', '2023-07-01', '2023-07-15'],
    'transaction_amount': [250.0, -150.0, 0.0, 420.5, 300.0, 500.0, -
20.0]
}
df = pd.DataFrame(data)

print("----------- Original DataFrame -----------")
print(df)

# 1. Convert transaction_date to proper datetime format
df['transaction_date'] = pd.to_datetime(df['transaction_date'])

# 2. Create a new column for “Month-Year”
df['Month-Year'] = df['transaction_date'].dt.strftime('%Y-%m')

# 3. Remove rows with negative or zero transaction amounts
df = df[df['transaction_amount'] > 0].reset_index(drop=True)

# 4. Normalize the "transaction_amount" column using Min-Max scaling
min_amount = df['transaction_amount'].min()
max_amount = df['transaction_amount'].max()
df['transaction_amount_normalized'] = (df['transaction_amount'] -
min_amount) / (max_amount - min_amount)

print("\n----------- Preprocessed DataFrame -----------")
```

```
print(df)
```

OUTPUT :

```
---------- Original DataFrame ----------
   transaction_id transaction_date  transaction_amount
0             101       2023-05-21               250.0
1             102       2023-05-22              -150.0
2             103       2023-06-01                0.0
3             104       2023-06-05               420.5
4             105       2023-06-10               300.0
5             106       2023-07-01               500.0
6             107       2023-07-15               -20.0

---------- Preprocessed DataFrame ----------
   transaction_id transaction_date  transaction_amount Month-Year  \
0             101       2023-05-21               250.0    2023-05
1             104       2023-06-05               420.5    2023-06
2             105       2023-06-10               300.0    2023-06
3             106       2023-07-01               500.0    2023-07

   transaction_amount_normalized
0                          0.000
1                          0.682
2                          0.200
3                          1.000
```

CODE EXPLANATION :

- **Convert Date Format: The transaction_date column, which contains text, is converted into a standardized datetime format using pd.to_datetime(). This allows for proper date-based calculations.**
- **Create Month-Year Column: A new column, Month-Year, is created by extracting the year and month from the transaction_date. The dt.strftime('%Y-%m') function formats it as YYYY-MM.**
- **Remove Invalid Rows: The script filters the DataFrame to keep only the rows where the transaction_amount is greater than zero. This effectively removes records for returns (negative amounts) or zero-value transactions.**
- **Normalize Transaction Amount: The transaction_amount is rescaled to a range between 0 and 1 using Min-Max scaling. This is a common preprocessing step for machine learning algorithms and is done by applying the formula (value - min) / (max - min) to each amount.**

PROMPT 03 :

Please write a Python script to clean up a patient records dataset. It should fill in missing numbers for blood pressure and heart rate using the average, convert heights from centimeters to meters, make the gender labels consistent (e.g., 'M' becomes 'Male'), and get rid of the patient ID column

CODE :

```
import pandas as pd
import numpy as np

# Create a sample healthcare DataFrame
data = {
    'patient_id': [1, 2, 3, 4, 5, 6],
    'gender': ['M', 'Female', 'male', 'F', 'Male', 'M'],
    'blood_pressure': [120, 130, np.nan, 110, 125, np.nan],
    'heart_rate': [80, np.nan, 75, 85, np.nan, 90],
    'height_cm': [175, 160, 180, 170, 185, 165]
}
df = pd.DataFrame(data)

print("----------- Original DataFrame -----------")
print(df)

# 1. Fill missing numeric values with the column mean
for col in ['blood_pressure', 'heart_rate']:
    mean_value = df[col].mean()
    df[col].fillna(mean_value, inplace=True)

# 2. Standardize units (convert height from cm to meters)
df['height_m'] = df['height_cm'] / 100

# 3. Correct inconsistent categorical labels
gender_mapping = {
    'M': 'Male',
    'male': 'Male',
    'F': 'Female',
    'Female': 'Female'
}
df['gender'] = df['gender'].replace(gender_mapping)

# 4. Drop irrelevant and redundant columns
df_cleaned = df.drop(columns=['patient_id', 'height_cm'])
```

```
print("\n---------- Cleaned DataFrame ----------")
print(df_cleaned)
```

CODE EXPLANATION :

- Fill Missing Values: The script first calculates the mean (average) for the blood_pressure and heart_rate columns. It then fills any missing values (NaN) in these columns with their respective means.
- Standardize Units: A new column, height_m, is created by dividing all values in the height_cm column by 100, effectively converting the height measurement from centimeters to meters.
- Correct Labels: A dictionary (gender_mapping) is used to define standard labels. The replace() function is then used on the gender column to consolidate all variations like "M" and "male" into the single, consistent label "Male".
- Drop Columns: Finally, the original patient_id and height_cm columns are removed using drop(). The patient ID is often irrelevant for model training, and the height in cm is now redundant since it has been converted to meters.

PROMT 04 :

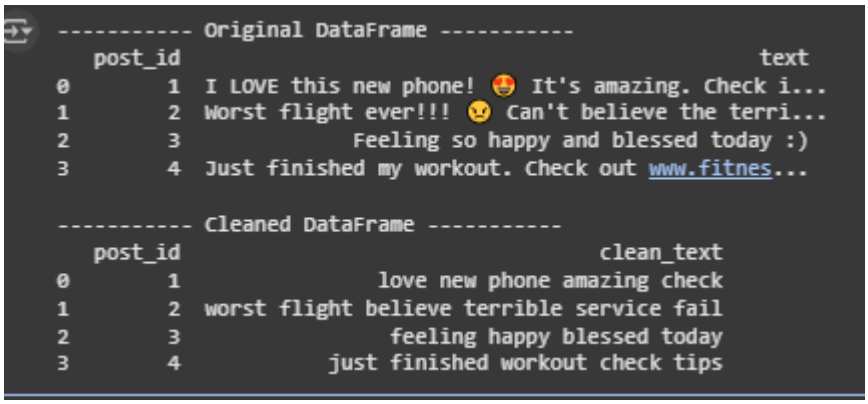
Could you write a Python script for me that cleans up social media posts? I need it to remove all the junk like links, emojis, and hashtags, convert everything to lowercase, and then get rid of common words like 'the' and 'a' so I can use the text for sentiment analysis

CODE :

```
import pandas as pd
import re
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS

# Create a sample social media DataFrame
data = {
```

```
    'post_id': [1, 2, 3, 4],
    'text': [
        "I LOVE this new phone! 😍 It's amazing. Check it out:
https://example.com",
        "Worst flight ever!!! 😫 Can't believe the terrible service.
#fail",
        "Feeling so happy and blessed today :)",
        "Just finished my workout. Check out www.fitness.com for tips."
    ]
}
df = pd.DataFrame(data)

print("----------- Original DataFrame -----------")
print(df)

# Define the set of stopwords
stop_words = set(ENGLISH_STOP_WORDS)

def clean_text(text):
    # 1. Remove special characters, URLs, and emojis
    text = re.sub(r'http\S+|www\.\S+', '', text)  # Remove URLs
    text = re.sub(r'[^a-zA-Z\s]', '', text)     # Remove emojis and
special characters

    # 2. Convert all text to lowercase
    text = text.lower()

    # 3. Tokenize and remove stopwords
    tokens = text.split()
    filtered_tokens = [word for word in tokens if word not in
stop_words]

    # Return cleaned text as a single string
    return ' '.join(filtered_tokens)

# Apply the cleaning function to the 'text' column
df['clean_text'] = df['text'].apply(clean_text)

print("\n----------- Cleaned DataFrame -----------")
print(df[['post_id', 'clean_text']])
```

OUTPUT :



```
---------- Original DataFrame ----------
   post_id                                               text
0        1  I LOVE this new phone! 😍 It's amazing. Check i...
1        2  Worst flight ever!!! 😠 Can't believe the terri...
2        3              Feeling so happy and blessed today :)
3        4  Just finished my workout. Check out www.fitnes...

---------- Cleaned DataFrame ----------
   post_id                                clean_text
0        1               love new phone amazing check
1        2  worst flight believe terrible service fail
2        3                feeling happy blessed today
3        4           just finished workout check tips
```

CODE EXPLANATION :

- **Remove Unwanted Elements**: The script uses regular expressions (re.sub) to find and remove URLs (starting with http or www) and any character that is not a letter or a space. This effectively cleans out emojis, punctuation, and hashtags.
- **Convert to Lowercase**: All text is converted to lowercase using .lower(). This ensures that words like "Love", "love", and "LOVE" are treated as the same word, which is crucial for accurate analysis.
- **Tokenization**: The cleaned text is broken down into a list of individual words (tokens) using .split().
- **Remove Stopwords**: The script filters out common English "stopwords" (e.g., "a", "the", "is", "in") from the token list. These words carry little meaning for sentiment analysis, and removing them helps the model focus on the important words. The final result is a clean string of meaningful words.

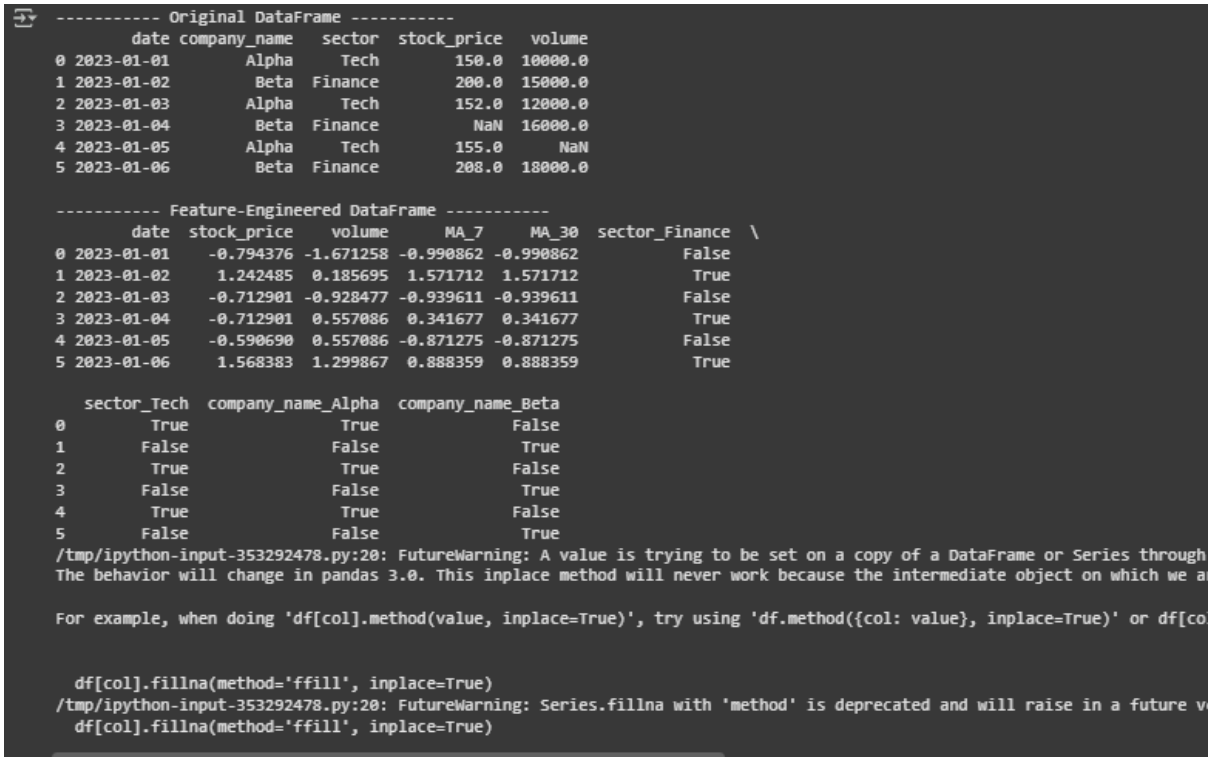
PROMPT 05 :

Could you write a Python script for financial data? I need to fill in some missing stock prices and volumes, then create new features like 7-day and 30-day moving averages for the price. After that, could you normalize the numerical data and convert the company and sector names into a format suitable for a machine learning model?

CODE :

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

# Create a sample financial DataFrame
data = {
    'date': pd.to_datetime(['2023-01-01', '2023-01-02', '2023-01-03',
'2023-01-04', '2023-01-05', '2023-01-06']),
    'company_name': ['Alpha', 'Beta', 'Alpha', 'Beta', 'Alpha',
'Beta'],
    'sector': ['Tech', 'Finance', 'Tech', 'Finance', 'Tech',
'Finance'],
    'stock_price': [150, 200, 152, np.nan, 155, 208],
    'volume': [10000, 15000, 12000, 16000, np.nan, 18000]
}
df = pd.DataFrame(data)

print("----------- Original DataFrame -----------")
print(df)

# 1. Handle missing values using forward fill
for col in ['stock_price', 'volume']:
    df[col].fillna(method='ffill', inplace=True)

# 2. Create new features (moving averages)
# Note: Grouping by company is crucial for correct calculation
df['MA_7'] = df.groupby('company_name')['stock_price'].transform(lambda
x: x.rolling(window=7, min_periods=1).mean())
df['MA_30'] =
df.groupby('company_name')['stock_price'].transform(lambda x:
x.rolling(window=30, min_periods=1).mean())

# 3. Normalize continuous variables using StandardScaler
scaler = StandardScaler()
continuous_cols = ['stock_price', 'volume', 'MA_7', 'MA_30']
df[continuous_cols] = scaler.fit_transform(df[continuous_cols])

# 4. Encode categorical columns
df_engineered = pd.get_dummies(df, columns=['sector', 'company_name'])

print("\n----------- Feature-Engineered DataFrame -----------")
print(df_engineered)
```

OUTPUT :



```
---------- Original DataFrame ----------
        date company_name   sector  stock_price   volume
0 2023-01-01        Alpha     Tech        150.0  10000.0
1 2023-01-02         Beta  Finance        200.0  15000.0
2 2023-01-03        Alpha     Tech        152.0  12000.0
3 2023-01-04         Beta  Finance          NaN  16000.0
4 2023-01-05        Alpha     Tech        155.0      NaN
5 2023-01-06         Beta  Finance        208.0  18000.0

---------- Feature-Engineered DataFrame ----------
        date  stock_price    volume      MA_7     MA_30  sector_Finance  \
0 2023-01-01    -0.794376 -1.671258 -0.990862 -0.990862           False
1 2023-01-02     1.242485  0.185695  1.571712  1.571712            True
2 2023-01-03    -0.712901 -0.928477 -0.939611 -0.939611           False
3 2023-01-04    -0.712901  0.557086  0.341677  0.341677            True
4 2023-01-05    -0.590690  0.557086 -0.871275 -0.871275           False
5 2023-01-06     1.568383  1.299867  0.888359  0.888359            True

   sector_Tech  company_name_Alpha  company_name_Beta
0         True                True              False
1        False               False               True
2         True                True              False
3        False               False               True
4         True                True              False
5        False               False               True
/tmp/ipython-input-353292478.py:20: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we a

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


  df[col].fillna(method='ffill', inplace=True)
/tmp/ipython-input-353292478.py:20: FutureWarning: Series.fillna with 'method' is deprecated and will raise in a future v
  df[col].fillna(method='ffill', inplace=True)
```

CODE EXPLANNATION :

- Handle Missing Values: Missing data in stock_price and volume are handled using ffill (forward fill). This method propagates the last known value forward, which is a common approach for time-series data like financial records.
- Create Moving Averages: Two new features are engineered: a 7-day (MA_7) and a 30-day (MA_30) moving average for the stock_price. The calculation is grouped by company_name to ensure the moving average is calculated independently for each company.
- Normalize Variables: StandardScaler from scikit-learn is used to normalize the continuous numerical columns. This process rescales the data to have a mean of 0 and a standard deviation of 1, which is a standard requirement for many machine learning models.
- Encode Categorical Columns: The text-based sector and company_name columns are converted into a numerical format using one-hot encoding (pd.get_dummies). This creates new binary columns for each unique category, making the data suitable for model training.