ASSIGNMENT - 17.4

K.VIKAS
2403A52126

Use AI to generate a Python script for cleaning an employee dataset.
Instructions:
• Handle missing values in columns (salary, department, joining_date).
• Convert the "joining_date" column into proper datetime format.
• Standardize department names (e.g., "HR", "hr", "Human Resources" → "HR").
• Encode categorical variables (department, job_role).
Expected Output:
• A cleaned Pandas DataFrame with consistent departments, proper dates, and encoded features.

Expected code :

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

# Generate sample data
data = {
    'name': [f'Employee {i}' for i in range(1, 31)],
    'employee id': [1001 + i for i in range(30)],
    'education qualification': ['BSc', 'MSc', 'PhD'] * 10,
    'years of experience': np.random.randint(1, 20, size=30),
    'salary': np.random.randint(30000, 120000, size=30),
    'department': ['IT', 'HR', 'Finance', 'Marketing', 'Sales'] * 6,
    'joining date': pd.to_datetime(pd.date_range(start='2010-01-01', periods=30, freq='M'))
}

df = pd.DataFrame(data)

# Introduce some missing values
for col in ['education qualification', 'years of experience', 'salary', 'department', 'joining date']:
    df.loc[df.sample(frac=0.1).index, col] = np.nan

display(df)
```

Dataset:

	name	employee id	education qualification	years of experience	salary	department	joining date
0	Employee 1	1001	BSc	18.0	70690.0	IT	2010-01-31
1	Employee 2	1002	MSc	12.0	85842.0	HR	2010-02-28
2	Employee 3	1003	PhD	1.0	91944.0	Finance	NaT
3	Employee 4	1004	BSc	2.0	102795.0	NaN	2010-04-30
4	Employee 5	1005	NaN	13.0	87078.0	Sales	2010-05-31
5	Employee 6	1006	PhD	6.0	102139.0	IT	2010-06-30
6	Employee 7	1007	BSc	NaN	101921.0	HR	2010-07-31
7	Employee 8	1008	NaN	13.0	61210.0	Finance	2010-08-31
8	Employee 9	1009	NaN	13.0	76914.0	Marketing	2010-09-30
9	Employee 10	1010	BSc	17.0	99374.0	Sales	2010-10-31
10	Employee 11	1011	MSc	7.0	44220.0	IT	2010-11-30
11	Employee 12	1012	PhD	15.0	68952.0	HR	2010-12-31
12	Employee 13	1013	BSc	8.0	80793.0	Finance	2011-01-31
13	Employee 14	1014	MSc	2.0	59398.0	Marketing	2011-02-28
14	Employee 15	1015	PhD	15.0	36346.0	Sales	2011-03-31
15	Employee 16	1016	BSc	17.0	36243.0	IT	2011-04-30
16	Employee 17	1017	MSc	3.0	NaN	HR	2011-05-31
17	Employee 18	1018	PhD	14.0	34971.0	Finance	2011-06-30
18	Employee 19	1019	BSc	8.0	38424.0	Marketing	2011-07-31
19	Employee 20	1020	MSc	5.0	75094.0	Sales	2011-08-31
20	Employee 21	1021	PhD	19.0	45957.0	IT	2011-09-30

19	Employee 20	1020	MSc	5.0	75094.0	Sales	2011-08-31
20	Employee 21	1021	PhD	19.0	45957.0	IT	2011-09-30
21	Employee 22	1022	BSc	3.0	50691.0	HR	2011-10-31
22	Employee 23	1023	MSc	13.0	53587.0	Finance	2011-11-30
23	Employee 24	1024	PhD	13.0	NaN	NaN	NaT
24	Employee 25	1025	BSc	15.0	42128.0	NaN	2012-01-31
25	Employee 26	1026	MSc	14.0	89388.0	IT	2012-02-29
26	Employee 27	1027	PhD	6.0	102386.0	HR	2012-03-31
27	Employee 28	1028	BSc	NaN	NaN	Finance	NaT
28	Employee 29	1029	MSc	5.0	65362.0	Marketing	2012-05-31
29	Employee 30	1030	PhD	NaN	45277.0	Sales	2012-06-30

Identifying and handling missing values:

```python
# 1. Identify columns with missing values and their extent
missing_values = df.isnull().sum()
print("Missing values per column:")
display(missing_values)

# Columns to consider for imputation/dropping
cols_with_missing = missing_values[missing_values > 0].index.tolist()

# 2. & 3. & 4. Handle missing values
for col in cols_with_missing:
    if df[col].isnull().sum() / len(df) * 100 < 10:  # Drop rows if missing percentage is small
        df.dropna(subset=[col], inplace=True)
        print(f"Dropped rows with missing values in column: {col}")
    elif df[col].dtype in ['int64', 'float64']:  # Impute numerical columns with mean
        mean_val = df[col].mean()
        df[col].fillna(mean_val, inplace=True)
        print(f"Imputed missing values in numerical column '{col}' with mean: {mean_val}")
    elif df[col].dtype == 'object':  # Impute categorical columns with mode
        mode_val = df[col].mode()[0]
        df[col].fillna(mode_val, inplace=True)
        print(f"Imputed missing values in categorical column '{col}' with mode: {mode_val}")
    elif df[col].dtype == '<M8[ns]': # Impute datetime columns with a suitable value, e.g., the m
        mode_val = df[col].mode()[0]
        df[col].fillna(mode_val, inplace=True)
        print(f"Imputed missing values in datetime column '{col}' with mode: {mode_val}")


# 5. Verify that there are no remaining missing values
print("\nMissing values after handling:")
display(df.isnull().sum())

display(df.head())
```

Missing values per column:

	0
name	0
employee id	0
education qualification	3
years of experience	3
salary	3
department	3
joining date	3

	0
name	0
employee id	0
education qualification	0
years of experience	0
salary	0
department	0
joining date	0

dtype: int64

	name	employee id	education qualification	years of experience	salary	department	joining date
0	Employee 1	1001	BSc	18.0	70690.0	IT	2010-01-31
1	Employee 2	1002	MSc	12.0	85842.0	HR	2010-02-28
2	Employee 3	1003	PhD	1.0	91944.0	Finance	2010-01-31
3	Employee 4	1004	BSc	2.0	102795.0	Finance	2010-04-30
4	Employee 5	1005	BSc	13.0	87078.0	Sales	2010-05-31

Converting joining date to proper date format:

```
df['joining date'] = pd.to_datetime(df['joining date'])
display(df.dtypes)
```

	0
name	object
employee id	int64
education qualification	object
years of experience	float64
salary	float64
department	object
joining date	datetime64[ns]

dtype: object

Standardize department names :

```
# 1. Inspect unique values
print("Unique values in 'department' before cleaning:")
display(df['department'].unique())

# 2. Convert to lowercase and remove leading/trailing whitespace
df['department'] = df['department'].str.lower().str.strip()

# 3. Replace any inconsistent spellings or variations with the correct, standardized department name.
# Based on the unique values above, there don't appear to be spelling inconsistencies, only case and potential whitespace.
# If there were, we would add a replace step here, e.g., df['department'] = df['department'].replace({'inconsistent': 'consistent'})

# Verify the changes
print("\nUnique values in 'department' after cleaning:")
display(df['department'].unique())
```

```
Unique values in 'department' before cleaning:
array(['IT', 'HR', 'Finance', 'Sales', 'Marketing'], dtype=object)

Unique values in 'department' after cleaning:
array(['it', 'hr', 'finance', 'sales', 'marketing'], dtype=object)
```

Encoding data:

```
# Apply one-hot encoding to 'education qualification' and 'department'
df_encoded = pd.get_dummies(df, columns=['education qualification', 'department'], dtype=int)

# Display the first few rows and data types of the modified DataFrame
display(df_encoded.head())
display(df_encoded.dtypes)
```

Expected output :

	name	employee id	years of experience	salary	joining date	education qualification_BSc	education qualification_MSc	education qualification_PhD	department_finance	department_hr	department_it
0	Employee 1	1001	18.0	70690.0	2010-01-31	1	0	0	0	0	1
1	Employee 2	1002	12.0	85842.0	2010-02-28	0	1	0	0	1	0
2	Employee 3	1003	1.0	91944.0	2010-01-31	0	0	1	1	0	0
3	Employee 4	1004	2.0	102795.0	2010-04-30	1	0	0	1	0	0
4	Employee 5	1005	13.0	87078.0	2010-05-31	1	0	0	0	0	0

department_marketing	department_sales
0	0
0	0
0	0
0	0
0	1

name	object
employee id	int64
years of experience	float64
salary	float64
joining date	datetime64[ns]
education qualification_BSc	int64
education qualification_MSc	int64
education qualification_PhD	int64
department_finance	int64
department_hr	int64
department_it	int64
department_marketing	int64
department_sales	int64

dtype: object

TASK 2:

Use AI to generate a script for preprocessing a sales transaction dataset.

Instructions:

• Convert transaction dates to proper date-time format.
• Create a new column for “Month-Year” from the transaction date.
• Remove rows with negative or zero transaction amounts.
• Normalize the "transaction_amount" column using Min-Max scaling.

Expected Output:

• A preprocessed DataFrame with valid dates, normalized amounts, and no invalid records.

Expected code and sample dataset :

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

# Create a sample sales transaction dataset
data = {
    'transaction_amount': np.random.rand(20) * 1000,  # Random amounts between 0 and 1000
    'transaction_date': pd.to_datetime(pd.date_range(start='2023-01-01', periods=20, freq='D')), # 20 consecutive dates
    'transaction_details': [f'Transaction {i+1}' for i in range(20)] # Sample transaction details
}

df_sales = pd.DataFrame(data)

print("Sample Sales Transaction Dataset:")
display(df_sales)
```

```
Sample Sales Transaction Dataset:
```

	transaction_amount	transaction_date	transaction_details
0	191.632182	2023-01-01	Transaction 1
1	969.217697	2023-01-02	Transaction 2
2	360.768200	2023-01-03	Transaction 3
3	617.373470	2023-01-04	Transaction 4
4	311.877877	2023-01-05	Transaction 5
5	322.095411	2023-01-06	Transaction 6
6	380.012783	2023-01-07	Transaction 7
7	172.828217	2023-01-08	Transaction 8
8	843.281760	2023-01-09	Transaction 9
9	573.041336	2023-01-10	Transaction 10
10	355.453219	2023-01-11	Transaction 11
11	309.862413	2023-01-12	Transaction 12
12	913.091955	2023-01-13	Transaction 13
13	602.946836	2023-01-14	Transaction 14
14	895.803058	2023-01-15	Transaction 15
15	367.338423	2023-01-16	Transaction 16
16	255.055797	2023-01-17	Transaction 17
17	276.112357	2023-01-18	Transaction 18
18	893.858447	2023-01-19	Transaction 19
19	387.519679	2023-01-20	Transaction 20

Converting date format and removing irrelevant columns :

```python
from sklearn.preprocessing import MinMaxScaler

# Ensure 'transaction_date' is in datetime format
df_sales['transaction_date'] = pd.to_datetime(df_sales['transaction_date'], errors='coerce')
print("Transaction dates converted to datetime.")

# Create 'Month-Year' column
df_sales['Month-Year'] = df_sales['transaction_date'].dt.strftime('%Y-%m')
print("'Month-Year' column created.")

# Remove rows with non-positive transaction amounts
initial_rows = df_sales.shape[0]
df_sales = df_sales[df_sales['transaction_amount'] > 0].copy() # Use .copy() to avoid SettingWithCopyWarning
removed_rows = initial_rows - df_sales.shape[0]
print(f"Removed {removed_rows} rows with non-positive 'transaction_amount'.")


# Normalize 'transaction_amount' using Min-Max scaling
if not df_sales.empty and 'transaction_amount' in df_sales.columns:
    scaler = MinMaxScaler()
    df_sales['transaction_amount_normalized'] = scaler.fit_transform(df_sales[['transaction_amount']])
    print("'transaction_amount' column normalized using Min-Max scaling.")
else:
    print("Cannot normalize 'transaction_amount': DataFrame is empty or column not found.")

# Display the preprocessed DataFrame
print("\nPreprocessed DataFrame:")
display(df_sales)
```

	transaction_amount	transaction_date	transaction_details	Month-Year	transaction_amount_normalized
0	191.632182	2023-01-01	Transaction 1	2023-01	0.023612
1	969.217697	2023-01-02	Transaction 2	2023-01	1.000000
2	360.768200	2023-01-03	Transaction 3	2023-01	0.235990
3	617.373470	2023-01-04	Transaction 4	2023-01	0.558201
4	311.877877	2023-01-05	Transaction 5	2023-01	0.174600
5	322.095411	2023-01-06	Transaction 6	2023-01	0.187430
6	380.012783	2023-01-07	Transaction 7	2023-01	0.260155
7	172.828217	2023-01-08	Transaction 8	2023-01	0.000000
8	843.281760	2023-01-09	Transaction 9	2023-01	0.841866
9	573.041336	2023-01-10	Transaction 10	2023-01	0.502534
10	355.453219	2023-01-11	Transaction 11	2023-01	0.229316
11	309.862413	2023-01-12	Transaction 12	2023-01	0.172069
12	913.091955	2023-01-13	Transaction 13	2023-01	0.929525
13	602.946836	2023-01-14	Transaction 14	2023-01	0.540086
14	895.803058	2023-01-15	Transaction 15	2023-01	0.907816
15	367.338423	2023-01-16	Transaction 16	2023-01	0.244240
16	255.055797	2023-01-17	Transaction 17	2023-01	0.103250
17	276.112357	2023-01-18	Transaction 18	2023-01	0.129690
18	893.858447	2023-01-19	Transaction 19	2023-01	0.905374
19	387.519679	2023-01-20	Transaction 20	2023-01	0.269581

TASK 3:

Use AI to generate a script for cleaning healthcare patient records.

Instructions:

• Fill missing values in numeric columns (e.g., blood_pressure, heart_rate) with column mean.

• Standardize units (convert height from cm to meters).

• Correct inconsistent categorical labels (e.g., "M", "Male", "male" → "Male").

• Drop irrelevant columns such as patient_id after cleaning.

Expected Output:

- **A cleaned healthcare dataset suitable for ML model training**

Expected and sample dataset :

```
import pandas as pd
import numpy as np

# Generate sample patient data
data = {
    'patient id': [101 + i for i in range(20)],
    'patient name': [f'Patient {i}' for i in range(1, 21)],
    'age': np.random.randint(20, 80, size=20),
    'gender': np.random.choice(['Male', 'Female'], size=20),
    'height in cm': np.random.randint(150, 190, size=20),
    'weight in kg': np.random.randint(50, 100, size=20),
    'heart rate': np.random.randint(60, 100, size=20),
    'blood pressure': [f'{sys} / {dia}' for sys, dia in zip(np.random.randint(90, 140, size=20), np.random.randint(60, 90, size=20))]
}

patient_df = pd.DataFrame(data)

# Introduce some missing values
for col in ['age', 'gender', 'height in cm', 'weight in kg', 'heart rate', 'blood pressure']:
    patient_df.loc[patient_df.sample(frac=0.15).index, col] = np.nan

display(patient_df)
```

	patient id	patient name	age	gender	height in cm	weight in kg	heart rate	blood pressure
0	101	Patient 1	38.0	Male	186.0	52.0	89.0	136 / 84
1	102	Patient 2	65.0	Male	168.0	NaN	NaN	136 / 76
2	103	Patient 3	62.0	Female	NaN	74.0	97.0	NaN
3	104	Patient 4	NaN	Female	168.0	NaN	82.0	NaN
4	105	Patient 5	46.0	NaN	NaN	89.0	79.0	132 / 86
5	106	Patient 6	NaN	Female	151.0	97.0	NaN	107 / 65
6	107	Patient 7	60.0	NaN	155.0	53.0	78.0	129 / 74
7	108	Patient 8	78.0	Male	185.0	78.0	NaN	NaN
8	109	Patient 9	23.0	Male	153.0	96.0	65.0	127 / 76
9	110	Patient 10	42.0	Female	161.0	70.0	81.0	106 / 77
10	111	Patient 11	NaN	Female	154.0	88.0	71.0	107 / 60
11	112	Patient 12	72.0	Female	155.0	79.0	77.0	113 / 67
12	113	Patient 13	32.0	Female	159.0	74.0	65.0	101 / 69
13	114	Patient 14	62.0	Female	172.0	90.0	83.0	94 / 61
14	115	Patient 15	54.0	Female	165.0	79.0	85.0	136 / 60
15	116	Patient 16	44.0	NaN	170.0	94.0	65.0	127 / 65
16	117	Patient 17	31.0	Male	NaN	67.0	66.0	116 / 78
17	118	Patient 18	76.0	Female	152.0	50.0	62.0	91 / 86
18	119	Patient 19	68.0	Female	181.0	61.0	87.0	94 / 68
19	120	Patient 20	77.0	Female	169.0	NaN	81.0	105 / 63

Handling and correcting missing values:

```python
# 1. Calculate and display the number of missing values for each column
missing_values = patient_df.isnull().sum()
print("Missing values per column:")
display(missing_values)

# 2. Determine which columns have missing values and their percentage
cols_with_missing = missing_values[missing_values > 0].index.tolist()
missing_percentage = (missing_values[cols_with_missing] / len(patient_df)) * 100
print("\nMissing value percentage per column:")
display(missing_percentage)

# 3. & 4. & 5. & 6. Handle missing values
for col in cols_with_missing:
    if missing_percentage[col] < 20:  # Using 20% as a threshold for dropping rows
        patient_df.dropna(subset=[col], inplace=True)
        print(f"Dropped rows with missing values in column: {col}")
    elif patient_df[col].dtype in ['int64', 'float64']:  # Impute numerical columns with mean
        mean_val = patient_df[col].mean()
        patient_df[col].fillna(mean_val, inplace=True)
        print(f"Imputed missing values in numerical column '{col}' with mean: {mean_val}")
    elif patient_df[col].dtype == 'object':  # Impute categorical/string columns with mode
        mode_val = patient_df[col].mode()[0]
        patient_df[col].fillna(mode_val, inplace=True)
        print(f"Imputed missing values in column '{col}' with mode: {mode_val}")

# 7. Verify that there are no remaining missing values
print("\nMissing values after handling:")
display(patient_df.isnull().sum())

# 8. Display the first few rows
display(patient_df)
```

patient id	0
patient name	0
age	0
gender	0
height in cm	0
weight in kg	0
heart rate	0
blood pressure	0

dtype: int64

	patient id	patient name	age	gender	height in cm	weight in kg	heart rate	blood pressure
0	101	Patient 1	38.0	Male	1.86	52.0	89.0	136 / 84
8	109	Patient 9	23.0	Male	1.53	96.0	65.0	127 / 76
9	110	Patient 10	42.0	Female	1.61	70.0	81.0	106 / 77
11	112	Patient 12	72.0	Female	1.55	79.0	77.0	113 / 67
12	113	Patient 13	32.0	Female	1.59	74.0	65.0	101 / 69
13	114	Patient 14	62.0	Female	1.72	90.0	83.0	94 / 61
14	115	Patient 15	54.0	Female	1.65	79.0	85.0	136 / 60
17	118	Patient 18	76.0	Female	1.52	50.0	62.0	91 / 86
18	119	Patient 19	68.0	Female	1.81	61.0	87.0	94 / 68

Converting patient height to meters and standardise gender values :

```
patient_df['height in cm'] = patient_df['height in cm'] / 100
display(patient_df.head())
```

	patient id	patient name	age	gender	height in cm	weight in kg	heart rate	blood pressure
0	101	Patient 1	38.0	Male	1.86	52.0	89.0	136 / 84
8	109	Patient 9	23.0	Male	1.53	96.0	65.0	127 / 76
9	110	Patient 10	42.0	Female	1.61	70.0	81.0	106 / 77
11	112	Patient 12	72.0	Female	1.55	79.0	77.0	113 / 67
12	113	Patient 13	32.0	Female	1.59	74.0	65.0	101 / 69

```
# 1. Print the unique values in the 'gender' column
print("Unique values in 'gender' before cleaning:")
display(patient_df['gender'].unique())

# 2. If there are inconsistencies, use the .replace() method to standardize the values.
# Based on the unique values, we will standardize 'Male' to 'Male' and 'Female' to 'Female'.
# This step is included for completeness, even if the initial unique values are already consistent.
patient_df['gender'] = patient_df['gender'].replace({'Male ': 'Male', 'Female ': 'Female'})

# 3. Print the unique values in the 'gender' column again to verify
print("\nUnique values in 'gender' after cleaning:")
display(patient_df['gender'].unique())
```

```
Unique values in 'gender' before cleaning:
array(['Male', 'Female'], dtype=object)

Unique values in 'gender' after cleaning:
array(['Male', 'Female'], dtype=object)
```

DataFrame head after dropping columns:

	age	gender	height in cm	weight in kg	heart rate	blood pressure
0	38.0	Male	1.86	52.0	89.0	136 / 84
8	23.0	Male	1.53	96.0	65.0	127 / 76
9	42.0	Female	1.61	70.0	81.0	106 / 77
11	72.0	Female	1.55	79.0	77.0	113 / 67
12	32.0	Female	1.59	74.0	65.0	101 / 69

TASK 4 :

Use AI to write a script to preprocess a social media text dataset.

Instructions:

• Remove special characters, URLs, and emojis from text.

• Convert all text to lowercase.

• Tokenize and remove stopwords.

• Apply lemmatization for standardizing words.

Expected Output:

• A processed dataset with clean text, ready for NLP sentiment analysis.

Expected code :

```
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download necessary NLTK data
nltk.download('stopwords')
nltk.download('wordnet')

# Create a sample dataset
data = {'text': ["Hello world! This is a test post with a URL https://example.com and an emoji 😊.",
                 "Another post with #hashtags and @mentions.",
                 "Testing 123... Removing special characters!",
                 "  Leading and trailing spaces and multiple   spaces inside.  ",
                 "This is a duplicate post for testing purposes.",
                 "this is a duplicate post for testing purposes.",
                 "Let's see if lemmatization works for running."]}
df = pd.DataFrame(data)

# Initialize stop words and lemmatizer
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
```

```
    def preprocess_text(text):
        # Remove URLs
        text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
        # Remove special characters and emojis
        text = re.sub(r'[^\w\s]', '', text)
        # Convert to lowercase
        text = text.lower()
        # Tokenize (simple split by whitespace)
        tokens = text.split()
        # Remove stopwords and lemmatize
        clean_tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
        return ' '.join(clean_tokens)

    # Apply preprocessing
    df['clean_text'] = df['text'].apply(preprocess_text)

    # Display the result
    display(df)
```

Preprocessed data :

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

	text	clean_text
0	Hello world! This is a test post with a URL ht...	hello world test post url emoji
1	Another post with #hashtags and @mentions.	another post hashtags mention
2	Testing 123... Removing special characters!	testing 123 removing special character
3	Leading and trailing spaces and multiple s...	leading trailing space multiple space inside
4	This is a duplicate post for testing purposes.	duplicate post testing purpose
5	this is a duplicate post for testing purposes.	duplicate post testing purpose
6	Let's see if lemmatization works for running.	let see lemmatization work running

TASK 5 :
Use AI to create a preprocessing script for a financial dataset.
Instructions:
• Handle missing values in stock price and volume.
• Create new features such as moving average (7-day, 30-day).
• Normalize continuous variables using StandardScaler.
• Encode categorical columns (sector, company_name).
Expected Output:
• A feature-engineered DataFrame with new indicators and normalized values for ML tasks.

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

# Create a sample dataset
data = {
    'date': pd.to_datetime(['2023-01-01', '2023-01-02', '2023-01-03', '2023-01-04', '2023-01-05',
                            '2023-01-06', '2023-01-07', '2023-01-08', '2023-01-09', '2023-01-10',
                            '2023-01-11', '2023-01-12', '2023-01-13', '2023-01-14', '2023-01-15',
                            '2023-01-16', '2023-01-17', '2023-01-18', '2023-01-19', '2023-01-20',
                            '2023-01-21', '2023-01-22', '2023-01-23', '2023-01-24', '2023-01-25',
                            '2023-01-26', '2023-01-27', '2023-01-28', '2023-01-29', '2023-01-30']),
    'company_name': ['A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B',
                     'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B',
                     'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B'],
    'sector': ['Tech', 'Tech', 'Tech', 'Tech', 'Tech', 'Healthcare', 'Healthcare', 'Healthcare', 'Healthcare', 'Healthcare',
               'Tech', 'Tech', 'Tech', 'Tech', 'Tech', 'Healthcare', 'Healthcare', 'Healthcare', 'Healthcare', 'Healthcare',
               'Tech', 'Tech', 'Tech', 'Tech', 'Tech', 'Healthcare', 'Healthcare', 'Healthcare', 'Healthcare', 'Healthcare'],
    'stock_price': [150.0, 151.5, 152.0, np.nan, 153.5, 80.0, 81.0, 80.5, 82.0, 81.5,
                    154.0, 155.0, 154.5, 156.0, 157.0, 82.5, 83.0, np.nan, 84.0, 83.5,
                    158.0, 159.0, 160.0, 161.0, 162.0, 84.5, 85.0, 85.5, 86.0, 86.5],
    'volume': [10000, 10500, 11000, 11200, 11500, 5000, 5100, np.nan, 5300, 5200,
               11800, 12000, 12200, 12500, 12800, 5400, 5500, 5600, np.nan, 5800,
               13000, 13200, 13500, 13800, 14000, 5900, 6000, 6100, 6200, 6300]
}
df = pd.DataFrame(data)

# Sort by date for moving average calculation
df = df.sort_values(by='date')

# Handle missing values (using forward fill for simplicity, mean or other methods can be used)
df['stock_price'] = df['stock_price'].fillna(method='ffill')
df['volume'] = df['volume'].fillna(method='ffill')
```

```
# Create new features: moving averages
df['moving_average_7'] = df['stock_price'].rolling(window=7).mean()
df['moving_average_30'] = df['stock_price'].rolling(window=30).mean()

# Handle NaN values introduced by rolling window (fill with the first valid value)
df['moving_average_7'] = df['moving_average_7'].fillna(method='bfill')
df['moving_average_30'] = df['moving_average_30'].fillna(method='bfill')

# Normalize continuous variables
continuous_cols = ['stock_price', 'volume', 'moving_average_7', 'moving_average_30']
scaler = StandardScaler()
df[continuous_cols] = scaler.fit_transform(df[continuous_cols])

# Encode categorical columns
label_encoder_company = LabelEncoder()
df['company_name_encoded'] = label_encoder_company.fit_transform(df['company_name'])

label_encoder_sector = LabelEncoder()
df['sector_encoded'] = label_encoder_sector.fit_transform(df['sector'])

# Display the final DataFrame
display(df)
```

Sample dataset with handling missing values and normalise continuous variables:

	date	company_name	sector	stock_price	volume	moving_average_7	moving_average_30	company_name_encoded	sector_encoded
0	2023-01-01	A	Tech	0.840042	0.321517	0.757092	0.0	0	1
1	2023-01-02	A	Tech	0.881311	0.468105	0.757092	0.0	0	1
2	2023-01-03	A	Tech	0.895067	0.614694	0.757092	0.0	0	1
3	2023-01-04	A	Tech	0.895067	0.673329	0.757092	0.0	0	1
4	2023-01-05	A	Tech	0.936335	0.761282	0.757092	0.0	0	1
5	2023-01-06	B	Healthcare	-1.085819	-1.144366	0.757092	0.0	1	0
6	2023-01-07	B	Healthcare	-1.058306	-1.115049	0.757092	0.0	1	0
7	2023-01-08	B	Healthcare	-1.072063	-1.115049	-0.041157	0.0	1	0
8	2023-01-09	B	Healthcare	-1.030794	-1.056413	-0.839405	0.0	1	0
9	2023-01-10	B	Healthcare	-1.044550	-1.085731	-1.649140	0.0	1	0
10	2023-01-11	A	Tech	0.950091	0.849235	-1.626169	0.0	0	1
11	2023-01-12	A	Tech	0.977604	0.907871	-1.608940	0.0	0	1
12	2023-01-13	A	Tech	0.963847	0.966506	-0.753263	0.0	0	1
13	2023-01-14	A	Tech	1.005116	1.054459	0.108156	0.0	0	1
14	2023-01-15	A	Tech	1.032628	1.142412	0.986804	0.0	0	1
15	2023-01-16	B	Healthcare	-1.017038	-1.027096	0.992547	0.0	1	0
16	2023-01-17	B	Healthcare	-1.003282	-0.997778	1.009775	0.0	1	0
17	2023-01-18	B	Healthcare	-1.003282	-0.968460	0.194298	0.0	1	0
18	2023-01-19	B	Healthcare	-0.975769	-0.968460	-0.621179	0.0	1	0
19	2023-01-20	B	Healthcare	-0.989526	-0.909825	-1.436656	0.0	1	0
20	2023-01-21	A	Tech	1.060141	1.201047	-1.413685	0.0	0	1
21	2023-01-22	A	Tech	1.087653	1.259683	-1.390714	0.0	0	1
22	2023-01-23	A	Tech	1.115165	1.347636	-0.500580	0.0	0	1
23	2023-01-24	A	Tech	1.142677	1.435589	0.395296	0.0	0	1
24	2023-01-25	A	Tech	1.170190	1.494224	1.302658	0.0	0	1
25	2023-01-26	B	Healthcare	-0.962013	-0.880507	1.308400	0.0	1	0
26	2023-01-27	B	Healthcare	-0.948257	-0.851190	1.325629	0.0	1	0
27	2023-01-28	B	Healthcare	-0.934501	-0.821872	0.492923	0.0	1	0
28	2023-01-29	B	Healthcare	-0.920745	-0.792554	-0.345525	0.0	1	0
29	2023-01-30	B	Healthcare	-0.906989	-0.763237	-1.189716	0.0	1	0