AI ASSISTED CODING

LAB ASSIGNMENT – 17.4

HALL TICKET NO:2403A52394

BATCH NO : 14

TASK – 01:

```
import pandas as pd import numpy as np


# Create a dictionary with sample data data = {
'employee id': [101, 102, 103, 104, 105, 106, 107, 108,
109, 110],
'name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank', 'Grace', 'Heidi', 'Ivan', 'Judy'],
'age': [25, 30, np.nan, 35, 28, 40, 32, np.nan, 27, 38],
'dept': ['HR', 'IT', 'Sales', 'Marketing', 'IT', 'Finance', 'HR', 'Sales', 'Marketing', 'Finance'],
'pay scale': [50000, 60000, 55000, np.nan, 62000,
75000, 52000, 58000, np.nan, 70000],
```

```
'qualification': ['Bachelors', 'Masters', 'Bachelors', 'PhD', 'Masters', 'Bachelors',
'Masters', 'Bachelors', 'PhD', 'Masters'],

'joining_date': pd.to_datetime(['2020-01-15', '2019-05-

20', '2021-11-10', '2018-08-01', '2022-03-25', '2017-09-18',

'2020-06-30', '2019-12-01', '2021-07-22', '2018-04-12'])

}



# Create a pandas DataFrame employee_df = pd.DataFrame(data)


# Introduce some missing values employee_df.loc[[2, 7], 'dept'] = np.nan
employee_df.loc[[4, 9], 'qualification'] = np.nan


# Display the sample dataset print("Sample Employee Dataset:")
display(employee_df.head())


# Save the dataset to a CSV file employee_df.to_csv('employee_data.csv',
index=False)
```

print("\nSample dataset saved as 'employee_data.csv'") OUTPUT :

```
Sample Employee Dataset:
```

	employee id	name	age	dept	pay scale	qualification	joining_date
0	101	Alice	25.0	HR	50000.0	Bachelors	2020-01-15
1	102	Bob	30.0	IT	60000.0	Masters	2019-05-20
2	103	Charlie	NaN	NaN	55000.0	Bachelors	2021-11-10
3					NaN	PhD	2018-08-01
4	105	Eve	28.0	IT	62000.0	NaN	2022-03-25

Copy Improve

```
Sample dataset saved as 'employee_data.csv'
```

EXPLANATION :

Certainly! The code in cell 49e5ebec creates a sample employee dataset using the pandas library and saves it as a CSV file. Here's a breakdown:

1. **Import Libraries**: It imports pandas for data manipulation and numpy for numerical operations, specifically for creating NaN (Not a Number) values to represent missing data.

2. **Create Sample Data**: A Python dictionary

named data is created. This dictionary holds lists for each column of the dataset: 'employee id', 'name', 'age', 'dept', 'pay scale', 'qualification', and 'joining_date'. Notice that some values are np.nan, which will become missing values in the DataFrame. The 'joining_date' list is created by converting strings to datetime objects using pd.to_datetime.

3. **Create DataFrame**: The data dictionary is used to create a pandas DataFrame called employee_df.

4. **Introduce More Missing Values**: Additional missing values are intentionally introduced into the 'dept' and 'qualification' columns at specific row indices

using .loc and np.nan.

5. **Display Sample Dataset**: The code prints a header "Sample Employee Dataset:" and then displays the first few rows of

the employee_df using display(employee_df.head()).

6. **Save to CSV**: The created DataFrame employee_df is saved to a CSV file named 'employee_data.csv' using employee_df.to_csv(). index=False is used to prevent writing the DataFrame index as a column in the CSV file.

7. **Confirmation Message**: Finally, it prints a message confirming that the sample dataset has been saved as 'employee_data.csv'.

This code effectively generates a small, representative dataset with some missing values that can be used for practicing data cleaning and preprocessing techniques.

MISSING VALUES:

Fill missing 'age' with the mean employee_df['age'] =

employee_df['age'].fillna(employee_df['age'].mean())

```
# Fill missing 'pay scale' with the median

employee_df['pay scale'] = employee_df['pay scale'].fillna(employee_df['pay scale'].median())



# Fill missing 'qualification' with a constant value

employee_df['qualification'] = employee_df['qualification'].fillna('Unknown')



display(employee_df)
```

TASK -02:

```
import pandas as pd

from sklearn.preprocessing import MinMaxScaler



# ---- Step 1: Load the dataset ----

# Replace 'sales_data.csv' with your actual file name or path

try:

df = pd.read_csv("employee_data.csv") print("Successfully loaded employee_data.csv")

except FileNotFoundError:
```

```
print("Error: employee_data.csv not found. Please ensure the file is in the correct directory.")

df = None



if df is not None:

#	Step 2: Convert transaction dates to datetime

format ----

# Assuming 'joining_date' is the date column in employee_data.csv

if 'joining_date' in df.columns:

df['joining_date'] = pd.to_datetime(df['joining_date'], errors='coerce')



# Remove rows with invalid or missing dates in 'joining_date'

df = df.dropna(subset=['joining_date'])


# ---- Step 3: Create a new "Month-Year" column ---- df['joining_month_year'] =

df['joining_date'].dt.strftime('%b-%Y') # e.g., "Jan-2020" else:
```

```
print("Warning: 'joining_date' column not found.
Skipping date conversion and month-year column creation.")


# Step 4: Remove rows with missing 'pay scale'
(analogous to transaction amount) ----
# Assuming 'pay scale' is the column to filter on if 'pay scale' in df.columns:
initial_rows = len(df)
df.dropna(subset=['pay scale'], inplace=True) rows_removed = initial_rows - len(df)
if rows_removed > 0:
print(f"Removed {rows_removed} rows with missing 'pay scale'.")
else:
print("Warning: 'pay scale' column not found. Skipping removal of rows with missing values.")


# Step 5: Normalize the "pay scale" using Min-Max
Scaling ----
# Assuming 'pay scale' is the column to normalize
```

```
if 'pay scale' in df.columns and not df['pay scale'].isnull().any():

scaler = MinMaxScaler()

df['normalized_pay_scale'] = scaler.fit_transform(df[['pay scale']])

print("Successfully normalized 'pay scale' column.")

elif 'pay scale' in df.columns and df['pay scale'].isnull().any():

print("Warning: Skipping normalization of 'pay scale' due to remaining missing values.")

else:

print("Warning: 'pay scale' column not found. Skipping normalization.")



# ---- Final Output ----

print("\n Preprocessing complete! Here’s the preprocessed data preview:\n")

display(df.head())


# Optional: Save the cleaned dataset df.to_csv("employee_data_preprocessed.csv",

index=False)
```

print("\nPreprocessed dataset saved as 'employee_data_preprocessed.csv'")

else:

PRINT("\NPREPROCESSING COULD NOT BE COMPLETED DUE TO FILE LOADING ERROR.")

OUTPUT:

	employee id	name	age	dept	pay scale	qualification	joining_date	joining_month_year	normalized_pay_scale
0	101	Alice	25.0	HR	50000.0	Bachelors	2020-01-15	Jan-2020	0.00
1	102	Bob	30.0	IT	60000.0	Masters	2019-05-20	May-2019	0.40
2	103	Charlie	NaN	NaN	55000.0	Bachelors	2021-11-10	Nov-2021	0.20
4	105	Eve	28.0	IT	62000.0	NaN	2022-03-25	Mar-2022	0.48
5	106	Frank	40.0	Finance	75000.0	Bachelors	2017-09-18	Sep-2017	1.00

```
Preprocessed dataset saved as 'employee_data_preprocessed.csv'
```

EXPLANATION :

1. Load the dataset: It attempts to load the employee_data.csv file into a pandas DataFrame named df. It includes error handling in case the file is not found.

2. Convert to datetime: If the df is loaded successfully, it checks for a 'joining_date' column and converts it to datetime objects using pd.to_datetime. errors='coerce' will turn any unparseable dates into NaT (Not a Time). It then removes any rows where 'joining_date' is NaT.

3. Create Month-Year column: It extracts the month and year from the 'joining_date' column and creates a new column called 'joining_month_year' in the format "Month- Year" (e.g., "Jan-2020").

4. Remove missing 'pay scale': It checks for a 'pay scale' column and removes any rows that have missing values (NaN) in this column using dropna().

5. Normalize 'pay scale': If the 'pay scale' column exists and has no missing values after the previous step, it applies Min-Max scaling to normalize the values in this column. Min-Max scaling transforms the values to a range between 0 and 1. The normalized values are stored in a new column called 'normalized_pay_scale'.

6. Final Output: It prints a message indicating the preprocessing is complete, displays the head of the preprocessed DataFrame, and saves the cleaned DataFrame to a new CSV file named employee_data_preprocessed.csv.

This script effectively cleans and prepares the employee data for further analysis or modeling by handling dates, removing missing values in a key column, and normalizing a numerical feature.

TASK – 03:

Healthcare Patient Records Cleaning Script

import pandas as pd import numpy as np

```
# Example: Load dataset
# df = pd.read_csv("patient_records.csv")


# Sample DataFrame for demonstration data = {
'patient_id': [101, 102, 103, 104],
'height_cm': [170, 165, np.nan, 180],
'blood_pressure': [120, np.nan, 130, 125],
'heart_rate': [80, 85, np.nan, 90], 'gender': ['M', 'Male', 'female', 'F']
}
df = pd.DataFrame(data)


# ----Data Cleaning----                    ----------


# 1. Fill missing numeric values with column mean numeric_cols =
['blood_pressure', 'heart_rate', 'height_cm']
df[numeric_cols] = df[numeric_cols].apply(lambda x: x.fillna(x.mean()))


# 2. Convert height from cm to meters
```

```
df['height_m'] = df['height_cm'] / 100



# 3. Standardize gender labels
df['gender'] = df['gender'].str.strip().str.lower().replace({ 'm': 'Male', 'male': 'Male',
'f': 'Female', 'female': 'Female'
})



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



# ----Cleaned Dataset print("□■ Cleaned Healthcare Dataset:") print(df)
```

OUTPUT :

```
✅ Cleaned Healthcare Dataset:
   blood_pressure  heart_rate  gender  height_m
0           120.0        80.0    Male  1.700000
1           125.0        85.0    Male  1.650000
2           130.0        85.0  Female  1.716667
3           125.0        90.0  Female  1.800000
```

EXPLANATION :

1. **Import Libraries:** It imports pandas for data manipulation and numpy for numerical operations, specifically for handling NaN (Not a Number) values.

2. **Load or Sample Data:** It includes a commented-out line to load data from a CSV file named "patient_records.csv". For demonstration purposes, it creates a sample pandas DataFrame with patient information.

3. **Data Cleaning Section:** This section performs several cleaning steps:

    - **Fill Missing Numeric Values:** It identifies numeric columns

(blood_pressure, heart_rate, height_cm) and fills any missing values (NaN) in these columns with the mean of the respective column.

    - **Convert Height:** It creates a new

column height_m by converting the height from centimeters to meters.

    - **Standardize Gender Labels:** It standardizes the 'gender' column by removing leading/trailing whitespace, converting all entries to lowercase, and then replacing variations like 'm' and 'male' with 'Male', and 'f' and 'female' with 'Female'.

- **Drop Irrelevant Columns:** It removes the original 'patient_id' and 'height_cm' columns as they are no longer needed after creating 'height_m' and the patient ID is not used in the subsequent analysis.

4. **Display Cleaned Dataset:** Finally, it prints a message indicating the cleaned dataset and displays the resulting DataFrame.

In essence, the script takes raw patient data, handles missing values, standardizes units and categorical labels, and removes unnecessary columns to prepare the data for further analysis.

TASK – 04:

Social Media Sentiment Dataset Preprocessing Script

import pandas as pd import re

import nltk

from nltk.corpus import stopwords from nltk.tokenize import word_tokenize

from nltk.stem import WordNetLemmatizer

Set NLTK data directory

nltk.data.path.append('/root/nltk_data')

Download required NLTK resources (run once) nltk.download('punkt') nltk.download('stopwords') nltk.download('wordnet')

nltk.download('punkt_tab') # Download 'punkt_tab' resource

----Load Example Data ----------

df = pd.read_csv("social_media_posts.csv")

data = { 'post': [

"I love this product! ●v-ˇ Check it out: https://example.com",

"Worst experience ever!!! #disappointed", "It’s okay... not great, not terrible ˙.³*숑*˙±±’˙˙.^˙˙‘", "Totally worth the money ,FCØ‘골곤ˇˇ躇蹠"

]

}

```
df = pd.DataFrame(data)


#    ----Text Preprocessing                                    ------------------------


# 1. Function to clean text def clean_text(text):
text = re.sub(r"http\S+|www\S+", "", text)    #
Remove URLs
text = re.sub(r"[^a-zA-Z\s]", "", text)  # Remove special chars, emojis
text = text.lower()       # Convert to lowercase
return text


# 2. Apply cleaning
df['clean_text'] = df['post'].apply(clean_text)


# 3. Tokenization
df['tokens'] = df['clean_text'].apply(word_tokenize)


# 4. Remove stopwords
```

```
stop_words = set(stopwords.words('english'))

df['tokens'] = df['tokens'].apply(lambda x: [w for w in x if w not in stop_words])



# 5. Lemmatization

lemmatizer = WordNetLemmatizer()

df['lemmatized'] = df['tokens'].apply(lambda x: [lemmatizer.lemmatize(w) for w in x])



#  ----Final Output------                                  ----------------------

print(" ■ Preprocessed Social Media Dataset:") print(df[['post', 'lemmatized']])
```

OUTPUT:

```
✅ Preprocessed Social Media Dataset:
                                                  post  \
0  I love this product! 😍 Check it out: https://e...
1             Worst experience ever!!! #disappointed
2          It’s okay... not great, not terrible 🤷
3                        Totally worth the money 💸 👍

                                lemmatized
0                   [love, product, check]
1  [worst, experience, ever, disappointed]
2                  [okay, great, terrible]
3                  [totally, worth, money]
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

EXPLANATION:

1. **Import Libraries**: It imports necessary

libraries: pandas for data manipulation, re for regular expressions (used for cleaning text), and nltk for natural language processing tasks like tokenization and lemmatization.

2. **NLTK Data Setup**: It attempts to download specific NLTK resources (punkt, stopwords, wordnet,

and punkt_tab). These resources are needed for tokenization, removing common words (stopwords), and reducing words to their base form (lemmatization).

3. **Load Example Data**: It creates a small example pandas DataFrame with a 'post' column containing

sample social media text. In a real scenario, you would likely load data from a file (like a CSV) using the commented-out line df = pd.read_csv("social_media_posts.csv").

4. **Text Preprocessing Functions**:
    - clean_text(text): This function takes a string as input and performs several cleaning steps:
        - Removes URLs using regular expressions.
        - Removes special characters and emojis, keeping only letters and spaces.
        - Converts the text to lowercase.
    - These cleaning steps help standardize the text and remove noise.
5. **Apply Cleaning**: It applies the clean_text function to the 'post' column and stores the cleaned text in a new column called 'clean_text'.
6. **Tokenization**: It uses nltk.word_tokenize to split the cleaned text into individual words (tokens) and stores them in a new column called 'tokens'.
7. **Remove Stopwords**: It defines a set of English stopwords and then filters the 'tokens' list for each post, removing common words that typically don't carry much sentiment (like 'the', 'a', 'is'). The result is stored back in the 'tokens' column.

8. **Lemmatization**: It uses nltk.WordNetLemmatizer to reduce each word in the 'tokens' list to its base or dictionary form (e.g., "running" becomes "run"). The result is stored in a new column called 'lemmatized'.

9. **Final Output**: Finally, it prints the original 'post' and the 'lemmatized' columns of the DataFrame, showing the result of the preprocessing steps.

In essence, the script takes raw social media text, cleans it up, breaks it into meaningful words, removes common words, and reduces words to their base form, making it ready for further analysis.

TASK – 05

```
import pandas as pd import numpy as np

from sklearn.preprocessing import StandardScaler,

LabelEncoder

# Sample data

df = pd.DataFrame({

'company_name': ['ABC Corp']*5 + ['XYZ Ltd']*5, 'sector': ['Tech']*5 + ['Finance']*5,

'stock_price': [100, 102, np.nan, 105, 107, 200, 202,

np.nan, 205, 208],
```

```
'volume': [1000, 1100, 1050, np.nan, 1150, 2000,

np.nan, 2100, 2200, 2250]

})



# Handle missing values

df['stock_price'].fillna(df['stock_price'].mean(), inplace=True)

df['volume'].fillna(df['volume'].mean(), inplace=True)


# Moving averages df['ma_7'] =

df.groupby('company_name')['stock_price'].transform(lam bda x:
x.rolling(7,1).mean())

df['ma_30'] = df.groupby('company_name')['stock_price'].transform(lam bda x:
x.rolling(30,1).mean())



# Normalize

scaler = StandardScaler()

df[['stock_price', 'volume', 'ma_7', 'ma_30']] = scaler.fit_transform(df[['stock_price',
'volume', 'ma_7', 'ma_30']])
```

```
# Encode categories

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

df['company_name'] = LabelEncoder().fit_transform(df['company_name'])



print(df)
```

OUTPUT:

```
   company_name  sector  stock_price    volume      ma_7     ma_30
0             0       1    -1.194150 -1.262148 -1.212725 -1.212725
1             0       1    -1.149613 -1.053959 -1.189328 -1.189328
2             0       1     0.000000 -1.158053 -0.778895 -0.778895
3             0       1    -1.082807  0.000000 -0.858106 -0.858106
4             0       1    -1.038270 -0.949864 -0.896273 -0.896273
5             1       0     1.032703  0.819746  1.127032  1.127032
6             1       0     1.077240  0.000000  1.150429  1.150429
7             1       0     0.000000  1.027935  0.780943  0.780943
8             1       0     1.144046  1.236124  0.896712  0.896712
9             1       0     1.210851  1.340219  0.980212  0.980212
/tmp/ipython-input-3203884722.py:14: 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


  df['stock_price'].fillna(df['stock_price'].mean(), inplace=True)
/tmp/ipython-input-3203884722.py:15: 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


  df['volume'].fillna(df['volume'].mean(), inplace=True)
```

EXPLANATION :

1. **Import Libraries**: Imports necessary libraries like pandas for data manipulation, numpy for numerical operations,

and StandardScaler and LabelEncoder from sklearn.p reprocessing for data scaling and encoding.

2. **Sample Data Creation**: Creates a sample pandas DataFrame df with columns for date, company name, sector, stock price, and volume. This is for demonstration purposes; in a real scenario, you

would load data from a file

(e.g., pd.read_csv("financial_data.csv")).

3. **Handle Missing Values**: Fills missing values (np.nan) in the 'stock_price' and 'volume' columns with the mean of their respective columns.

4. **Create Moving Average Features**:

    - Calculates the 7-day moving average (ma_7) of 'stock_price' for each company

using groupby('company_name') and rolling(wind ow=7,

min_periods=1).mean(). min_periods=1 ensures that the calculation starts as soon as one data point is available within the window.

    - Calculates the 30-day moving average (ma_30) similarly.

5. **Normalize Continuous Variables**:

    - Initializes a StandardScaler object.
    - Applies the StandardScaler to the continuous columns ('stock_price', 'volume', 'ma_7', 'ma_30') to normalize them. This scales the data so that it has a mean of 0 and a standard deviation of 1, which is often beneficial for machine learning models. The scaled values are stored in new columns with a _scaled suffix.

6. **Encode Categorical Columns**:

- Initializes LabelEncoder objects for 'sector' and 'company_name'.
- Applies LabelEncoder to the 'sector' and 'company_name' columns to convert their categorical values into numerical labels. This is necessary because most machine learning algorithms require numerical input.

7. **Drop Date Column (Optional)**: The code includes a commented-out line df = df.drop(columns=['date']) which you can uncomment if the 'date' column is not needed for your modeling task.

8. **Print Feature-Engineered Data**: Prints the head of the modified DataFrame to show the results of the feature engineering steps.

In essence, this script prepares the raw financial data for use in a machine learning model by handling missing values, creating relevant time-series features (moving averages), scaling numerical features, and encoding categorical features.