LAB ASSIGNMENT-17

Task-1:

Prompt:

Create a Python script to clean an employee dataset: fill missing values (salary, department, joining_date), convert joining_date to datetime, standardize department names (e.g., 'HR', 'hr', 'Human Resources' → 'HR'), and encode categorical variables (department, job_role).

Code and Output:

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

# ------------------------------
# Step 1: Generate a Sample Employee Dataset
# ------------------------------
data = {
    'employee_id': [101, 102, 103, 104, 105, 106, 107, 108],
    'salary': [50000, np.nan, 75000, 60000, None, 80000, 55000, np.nan],
    'department': ['HR', 'hr', 'Human Resources', 'IT', 'Finance', 'hr', 'Sales', None],
    'joining_date': ['2020-05-10', None, '2019-07-22', '', '2021/03/15', '2022-11-30', '2020-08-01', '2021-12-12'],
    'job_role': ['Manager', 'Analyst', 'Developer', 'Clerk', 'Manager', 'Analyst', 'Developer', 'Clerk']
}

# Create DataFrame with missing values and inconsistent department names
df = pd.DataFrame(data)

# ------------------------------
# Step 2: Data Cleaning and Preprocessing
# ------------------------------

# Handle missing values:
# For 'salary', fill missing values with the median salary calculated from non-missing entries
df['salary'] = df['salary'].fillna(df['salary'].median())

# For 'department', fill missing values with "Unknown"
df['department'] = df['department'].fillna("Unknown")

# For 'joining_date', fill missing or empty strings with a default date (e.g., '2000-01-01')
df['joining_date'] = df['joining_date'].replace('', None).fillna("2000-01-01")

# Convert 'joining_date' column to datetime format (errors='coerce' will convert invalid formats to NaT)
```

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

# Standardize department names:
# Map various forms of HR to a single standardized value "HR"
def standardize_department(dept):
    dept_lower = str(dept).strip().lower()
    if dept_lower in ['hr', 'human resources']:
        return "HR"
    return dept  # Return the original if no standardization rule applies

df['department'] = df['department'].apply(standardize_department)

# Encode categorical variables:
# Using one-hot encoding for 'department' and 'job_role'
cleaned_df = pd.get_dummies(df, columns=['department', 'job_role'], drop_first=True)

# ------------------------------
# Step 3: Output the Cleaned DataFrame
# ------------------------------
print(cleaned_df.head())
```

```
>>> %Run -c $EDITOR_CONTENT
   employee_id   salary  ... job_role_Developer  job_role_Manager
0          101  50000.0  ...              False              True
1          102  60000.0  ...              False             False
2          103  75000.0  ...               True             False
3          104  60000.0  ...              False             False
4          105  60000.0  ...              False              True

[5 rows x 10 columns]
```

Explanation:

The code performs the following key operations:

- **Data Generation:**
  Creates a sample employee dataset with various issues like missing values and inconsistent department names.
- **Missing Values Handling:**
  - Fills missing salaries with the median salary.
  - Replaces missing department entries with "Unknown".
  - Substitutes missing or empty `joining_date` values with a default date ("2000-01-01").
- **Data Type Conversion:**
  Converts the `joining_date` column into datetime objects to ensure consistent date formatting.
- **Department Standardization:**
  Uses a custom function to map different forms of "HR" (like "hr" or "Human Resources") to a single, standardized value ("HR").
- **Categorical Encoding:**
  Applies one-hot encoding to `department` and `job_role`, while dropping the first category to avoid multicollinearity.
- **Output:**
  Finally, prints the head of the cleaned DataFrame for a quick preview of the processed data.

Task 2:

Prompt:

Generate a Python script to preprocess a sales dataset: convert transaction_date to datetime, add Month-Year column, remove rows with non-positive transaction_amount, and apply Min-Max scaling to transaction_amount.

Code and Output:

```python
import pandas as pd

# ------------------------------
# Step 1: Generate a Sample Sales Transaction Dataset
# ------------------------------
data = {
    'transaction_id': [1, 2, 3, 4, 5, 6],
    'transaction_date': ['2025-01-15', '2025/02/20', '15-03-2025', '2025-04-05', '2025-05-10', '2025-06-25'],
    'transaction_amount': [250.0, -50.0, 0.0, 500.0, 750.0, 300.0]
}

df = pd.DataFrame(data)
# Step 2: Preprocess the Dataset

# Convert 'transaction_date' column to datetime format (invalid dates become NaT)
df['transaction_date'] = pd.to_datetime(df['transaction_date'], errors='coerce')

# Create a new column 'Month-Year' extracted from the 'transaction_date'
df['Month-Year'] = df['transaction_date'].dt.strftime('%b-%Y')

# Remove rows with negative or zero transaction amounts
df = df[df['transaction_amount'] > 0]

# Normalize the 'transaction_amount' column using Min-Max scaling without sklearn
min_amount = df['transaction_amount'].min()
max_amount = df['transaction_amount'].max()
df['transaction_amount'] = (df['transaction_amount'] - min_amount) / (max_amount - min_amount)

# Step 3: Output the Preprocessed DataFrame
print(df)
```

```
>>> %Run -c $EDITOR_CONTENT
   transaction_id transaction_date  transaction_amount Month-Year
0               1       2025-01-15                 0.0   Jan-2025
3               4       2025-04-05                 0.5   Apr-2025
4               5       2025-05-10                 1.0   May-2025
5               6       2025-06-25                 0.1   Jun-2025
>>>
```

Explanation:

1. **Dataset Creation**
   - Creates a dictionary with three keys:
     - `transaction_id`: Unique IDs for each transaction.
     - `transaction_date`: Dates of transactions in various formats.
     - `transaction_amount`: Monetary values for each transaction.
   - Converts this dictionary into a pandas DataFrame.
2. **Data Preprocessing**
   - **Convert Dates:**
     - Uses `pd.to_datetime` with `errors='coerce'` to convert the `transaction_date` column to datetime format.
     - Any invalid date strings are set to `NaT` (Not a Time).
   - **Extract Month-Year:**
     - Creates a new column `Month-Year` by extracting the month and year from the valid `transaction_date` values using the `dt.strftime('%b-%Y')` method.
   - **Filter Transactions:**
     - Removes rows where `transaction_amount` is less than or equal to zero, keeping only rows with a positive amount.
   - **Normalize Amounts:**
     - Applies Min-Max scaling to the `transaction_amount` column without using sklearn.
     - This is done by subtracting the minimum value and dividing by the range (max - min) so that all transaction amounts are scaled between 0 and 1.
3. **Output**
   - Finally, the cleaned and preprocessed DataFrame is printed.

Task 3:

Prompt:

Write a Python script to clean healthcare patient data: fill missing numeric values with column mean, convert height from cm to meters, standardize gender labels (e.g., 'M', 'Male', 'male' → 'Male'), and drop the patient_id column.

Code and Output:

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

# ------------------------------
# Generate a Sample Healthcare Patient Dataset
# ------------------------------
data = {
    'patient_id': [1, 2, 3, 4, 5, 6, 7, 8],
    'blood_pressure': [120, 130, np.nan, 110, 140, np.nan, 125, 135],
    'heart_rate': [80, np.nan, 70, 75, 90, 85, np.nan, 78],
    'height': [170, 165, 180, np.nan, 160, 175, 168, 172],  # Heights in cm
    'gender': ['M', 'Male', 'male', 'F', 'Female', 'f', 'F', 'female'],
    'age': [25, 35, 45, 55, np.nan, 65, 75, 85]  # Additional numeric column
}

df = pd.DataFrame(data)

# ------------------------------
# Data Cleaning and Preprocessing
# ------------------------------

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

# 2. Standardize units: Convert height from centimeters to meters
df['height'] = df['height'] / 100

# 3. Correct inconsistent categorical labels for 'gender'
def standardize_gender(g):
    g_clean = str(g).strip().lower()
```

```python
    if g_clean in ['m', 'male']:
        return 'Male'
    elif g_clean in ['f', 'female']:
        return 'Female'
    return g  # Return original if no rule applies

df['gender'] = df['gender'].apply(standardize_gender)

# 4. Drop irrelevant columns such as patient_id
df_cleaned = df.drop(columns=['patient_id'])

# ------------------------------
# Output the Cleaned Dataset
# ------------------------------
print("Cleaned Healthcare Patient Records:")
print(df_cleaned.head())
```

```
Cleaned Healthcare Patient Records:
   blood_pressure  heart_rate  height  gender   age
0      120.000000   80.000000    1.70    Male  25.0
1      130.000000   79.666667    1.65    Male  35.0
2      126.666667   70.000000    1.80    Male  45.0
3      110.000000   75.000000    1.70  Female  55.0
4      140.000000   90.000000    1.60  Female  55.0

>>>
```

Explanation:

1. **Import Libraries**
   The code starts by importing the necessary libraries:
   - `pandas` for handling data in tabular form.
   - `numpy` for handling numerical data and to use the `np.nan` value as missing data.
2. **Generate a Sample Healthcare Dataset**
   A dictionary called `data` is defined which contains sample patient records. Each key in the dictionary represents a column (like `patient_id`, `blood_pressure`, etc.) and the values are lists representing the data entries. This data is then converted into a pandas DataFrame called `df`.
3. **Fill Missing Values in Numeric Columns**
   - The code identifies numeric columns such as `blood_pressure`, `heart_rate`, `height`, and `age`.
   - It loops over these columns and fills any missing values (denoted by `np.nan`) with the mean value of that column. This ensures that there are no empty values in these fields.
4. **Standardize Units for Height**
   The `height` column is stored in centimeters. The code divides the values by 100 to convert them into meters, which might be the preferred unit for analysis or modeling.
5. **Correct Inconsistent Categorical Labels for Gender**
   - A function called `standardize_gender` is defined. It takes a gender value, converts it to lowercase, and then maps it to either `"Male"` or `"Female"`.
   - This function is applied to the `gender` column to standardize the labels, ensuring consistency in the data.
6. **Drop the Irrelevant `patient_id` Column**
   After cleaning the records, the `patient_id` column is removed from the DataFrame since it is assumed not to be useful for further analysis (e.g., in a machine learning model).
7. **Display the Cleaned Dataset**
   Finally, the cleaned DataFrame (`df_cleaned`) is printed to the console. This shows the first few rows of the cleaned data, which is now ready for further analysis or modeling.

Task 4:

Prompt:

Write a Python script to clean social media text data: remove special characters, URLs, and emojis; convert text to lowercase; tokenize and remove stopwords; and apply lemmatization to prepare for sentiment analysis.

Code & Output:

```python
import re
import pandas as pd

# Sample dataset: a list of social media text posts
data = {
    "text": [
        "Loving this new product! 😍 #amazing http://example.com",
        "Worst service ever!! 😡😡 Visit http://badservice.com for details!",
        "Happy days! Enjoying life & sunshine 😊 https://weather.com",
        "OMG! Can't believe it!!! #shocked 😱 http://trending.com",
        "Feeling blessed & grateful. Life is beautiful 😊💕"
    ]
}

df = pd.DataFrame(data)
print("Original Dataset:")
print(df)

# Define a simple stopword list (you can expand this list as needed)
stop_words = {
    "the", "and", "is", "in", "it", "for", "to", "a",
    "of", "this", "that", "with", "on", "at", "ever", "cant"
}

def clean_text(text):
    # Remove URLs
    text = re.sub(r'http\S+', '', text)
    # Remove emojis using an approximate Unicode regex pattern
    emoji_pattern = re.compile(
        "["
        u"\U0001F600-\U0001F64F"  # Emoticons
        u"\U0001F300-\U0001F5FF"  # Symbols & pictographs
```

```python
        u"\U0001F680-\U0001F6FF"  # Transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # Flags
        "]+", flags=re.UNICODE)
    text = emoji_pattern.sub(r'', text)
    # Remove special characters (keep letters, numbers, and whitespace)
    text = re.sub(r'[^\w\s]', '', text)
    # Convert to lowercase
    text = text.lower()
    return text

def simple_tokenize(text):
    # A very basic tokenizer that splits on whitespace
    return text.split()

def simple_lemmatize(word):
    # A basic and naive lemmatization:
    # Remove common suffixes: 'ing', 'ed', 'es' and trailing 's'
    if word.endswith('ing') and len(word) > 4:
        return word[:-3]
    if word.endswith('ed') and len(word) > 3:
        return word[:-2]
    if word.endswith('es') and len(word) > 3:
        return word[:-2]
    if word.endswith('s') and len(word) > 2:
        return word[:-1]
    return word

def preprocess_text(text):
    # Clean the text to remove URLs, emojis, special characters, and convert to lowercase
    cleaned = clean_text(text)
    # Tokenize the cleaned text
    tokens = simple_tokenize(cleaned)
```

```python
    # Remove stopwords
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # Apply simple lemmatization
    lemmatized_tokens = [simple_lemmatize(token) for token in filtered_tokens]
    # Return the processed text as a string
    return ' '.join(lemmatized_tokens)

# Apply preprocessing to each text entry in the dataset
df['processed_text'] = df['text'].apply(preprocess_text)

print("\nProcessed Dataset:")
print(df[['processed_text']])
```

```
>>> %Run -c $EDITOR_CONTENT
 Original Dataset:
                                                text
 0  Loving this new product! 😍 #amazing http://exa...
 1  Worst service ever!! 😡😡 Visit http://badservic...
 2  Happy days! Enjoying life & sunshine 😊 https:/...
 3  OMG! Can't believe it!!! #shocked 😱 http://tre...
 4   Feeling blessed & grateful. Life is beautiful 😊💕

 Processed Dataset:
                    processed_text
 0              lov new product amaz
 1         worst service visit detail
 2     happy day enjoy life sunshine
 3                omg believe shock
 4  feel bless grateful life beautiful

>>>
```

Explanation:

- **Importing libraries and data setup:**
  The script imports the required libraries:
  - `re` for working with regular expressions.
  - `pandas` for handling and manipulating the dataset.

  It then creates a sample dataset (a few social media posts) as a Python dictionary and converts it into a DataFrame. Finally, it prints the original dataset.
- **Defining stopwords:**
  A basic set of stopwords (common words that are typically removed) is defined. This helps to filter out words that may not be important for sentiment analysis.
- **Cleaning the text:**
  The `clean_text` function performs several cleaning steps:
  - **Remove URLs:** It uses a regex pattern to remove any web links.
  - **Remove emojis:** It applies a regex that targets emoji characters and removes them.
  - **Remove special characters:** It keeps only letters, numbers, and whitespace.
  - **Convert to lowercase:** It changes all characters to lowercase for uniformity.
- **Tokenization:**
  The `simple_tokenize` function splits the cleaned text into words by splitting on whitespace. This breaks the text into manageable tokens.
- **Simple lemmatization:**
  The `simple_lemmatize` function removes common suffixes like "ing", "ed", "es" and a trailing "s" from each word. This is a very basic method to reduce words to their base form (lemmatization).
- **Preprocessing text:**
  The `preprocess_text` function puts all these steps together:
  - It cleans the text using `clean_text`.
  - It tokenizes the text using `simple_tokenize`.
  - It filters out any stopwords from these tokens.
  - It applies the simple lemmatization to the remaining tokens.
  - Finally, it joins these tokens back into a single string which represents the cleaned text.
- **Applying preprocessing and printing:**
  The script applies the `preprocess_text` function on each entry in the original "text" column of the DataFrame, creating a new column called "processed_text." It then prints this processed dataset, which is ready for further NLP sentiment analysis.

Task 5:

Prompt:

Write a Python script to preprocess a financial dataset: fill missing values in stock price and volume, create 7-day and 30-day moving averages, normalize continuous columns with StandardScaler, and encode categorical columns like sector and company_name.

Code & Output:

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

# Generate a simulated financial dataset
np.random.seed(42)
num_days = 60
dates = pd.date_range(start='2025-01-01', periods=num_days, freq='D')
sectors = ['Technology', 'Finance', 'Healthcare']
companies = ['CompanyA', 'CompanyB', 'CompanyC']

data = {
    'date': dates,
    'stock_price': np.random.uniform(100, 200, size=num_days),
    'volume': np.random.randint(1000, 10000, size=num_days),
    'sector': np.random.choice(sectors, size=num_days),
    'company_name': np.random.choice(companies, size=num_days)
}
df = pd.DataFrame(data)

# Introduce missing values randomly in stock_price and volume
for col in ['stock_price', 'volume']:
    df.loc[df.sample(frac=0.1, random_state=42).index, col] = np.nan

# Handle missing values: use median imputation
for col in ['stock_price', 'volume']:
    median_val = df[col].median()
    df[col].fillna(median_val, inplace=True)

# Create new features: moving averages (7-day and 30-day) of stock_price.
df['MA_7'] = df['stock_price'].rolling(window=7, min_periods=1).mean()
df['MA_30'] = df['stock_price'].rolling(window=30, min_periods=1).mean()
```

```python

# List of continuous columns to normalize
cont_cols = ['stock_price', 'volume', 'MA_7', 'MA_30']

# Normalize continuous variables using manual StandardScaler approach
for col in cont_cols:
    mean_val = df[col].mean()
    std_val = df[col].std()
    df[col + '_norm'] = (df[col] - mean_val) / std_val

# Encode categorical columns using dummy variables
cat_cols = ['sector', 'company_name']
df_encoded = pd.get_dummies(df, columns=cat_cols, drop_first=True)

# Final feature engineered DataFrame
print("Feature engineered dataset:")
print(df_encoded.head())

# Optionally, save to CSV file for further ML tasks
df_encoded.to_csv("feature_engineered_financial_data.csv", index=False)
print("\nData saved to feature_engineered_financial_data.csv")
```

```
Feature engineered dataset:
        date  stock_price  ...  company_name_CompanyB  company_name_CompanyC
0 2025-01-01   147.562345  ...                   True                  False
1 2025-01-02   195.071431  ...                   True                  False
2 2025-01-03   173.199394  ...                  False                   True
3 2025-01-04   159.865848  ...                  False                  False
4 2025-01-05   115.601864  ...                  False                  False

[5 rows x 13 columns]

Data saved to feature_engineered_financial_data.csv

>>>
```

Explanation:

1. **Import Modules**
   The code imports necessary packages:
   - `pandas` for handling data as DataFrames
   - `numpy` for numerical operations
2. **Generate a Simulated Financial Dataset**
   - A date range of 60 days starting from January 1, 2025, is created.
   - Random data is generated for stock prices and trading volume.
   - Two categorical columns (`sector` and `company_name`) are generated from predefined lists.
   - All these columns are combined into a DataFrame.
3. **Introduce Missing Values**
   - The code randomly selects 10% of the rows for `stock_price` and `volume` columns, replacing those values with `NaN` (to simulate missing data).
4. **Handle Missing Values**
   - The missing values in `stock_price` and `volume` are filled with the median value of each respective column.
5. **Calculate Moving Averages**
   - Two new features are created:
     - `MA_7`: the 7-day moving average of the stock price.
     - `MA_30`: the 30-day moving average of the stock price.
   - The rolling mean is computed using a window of 7 or 30 days.
6. **Normalize Continuous Variables Without Sklearn**
   - For each continuous column (stock price, volume, and the moving averages), the script computes the mean and standard deviation.
   - A new column is added for each feature (appending `_norm`) that represents the standardized value, calculated as:
     (value - mean) / std
7. **Encode Categorical Columns**
   - The categorical columns (`sector` and `company_name`) are encoded using one-hot encoding (creating dummy variables).
   - The `drop_first=True` option is used to avoid creating redundant columns.
8. **Output the Final DataFrame**
   - The code prints the first few rows of the processed DataFrame, showing the new features and normalized columns.
   - The final dataset is saved as a CSV file named "feature_engineered_financial_data.csv" for later machine learning tasks.