**LAB ASSIGNMENT – 17**

**NAME :** V.NITHYA

**HALL.NO :** 2403A52346

**BATCH.NO :** AI 13

**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)

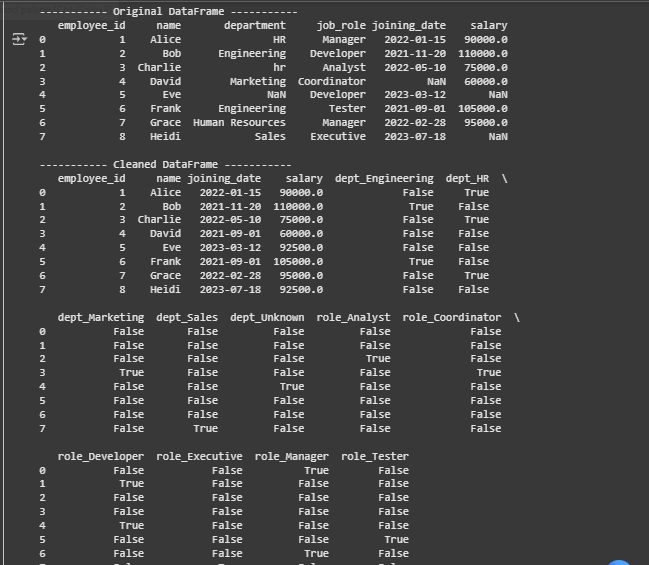
# 6. Encode categorical variables

df\_cleaned = pd.get\_dummies(df, columns=['department', 'job\_role'], prefix=['dept', 'role'])

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

print(df\_cleaned)

**OUTPUT :**

****

**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()

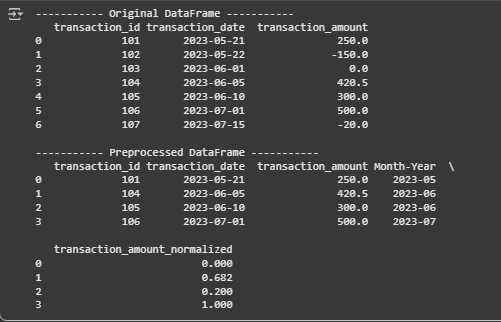
max\_amount = df['transaction\_amount'].max()

df['transaction\_amount\_normalized'] = (df['transaction\_amount'] - min\_amount) / (max\_amount - min\_amount)

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

print(df)

**OUTPUT :**

****

**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)

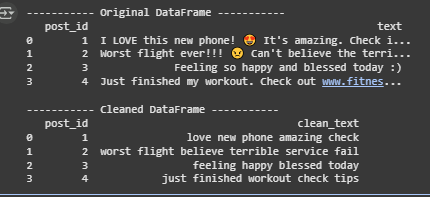
# Apply the cleaning function to the 'text' column

df['clean\_text'] = df['text'].apply(clean\_text)

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

print(df[['post\_id', 'clean\_text']])

**OUTPUT :**

****

**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 :**

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])

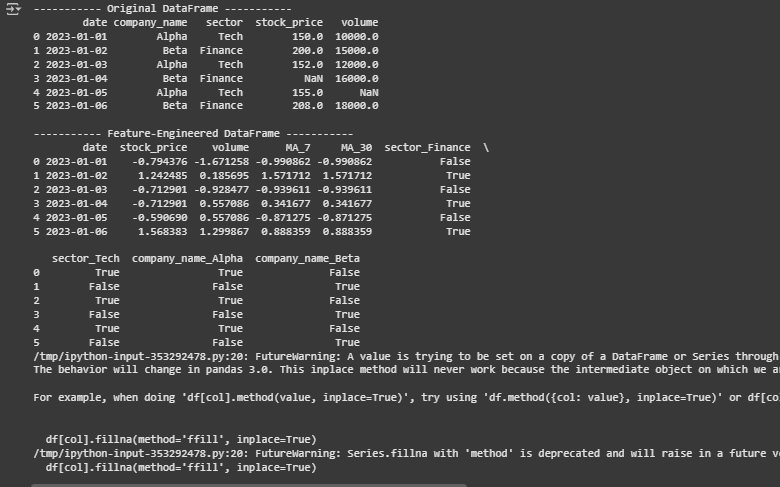
# 4. Encode categorical columns

df\_engineered = pd.get\_dummies(df, columns=['sector', 'company\_name'])

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

print(df\_engineered)

**OUTPUT :**

****

**CODE EXPLANATION :**

* 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.