Assignment-17

Task-1

Python Copy

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Load your dataset

# df = pd.read_csv('employee_data.csv')  # Uncomment and modify as needed


# Sample DataFrame for demonstration

data = {
    'salary': [50000, None, 70000, 60000, None],
    'department': ['HR', 'hr', 'Human Resources', 'Engineering', None],
    'joining_date': ['2020-01-15', '15/02/2021', None, 'March 3, 2022', '2021
    'job_role': ['Manager', 'Analyst', 'Manager', 'Engineer', 'Analyst']
}
df = pd.DataFrame(data)

# 1. Handle missing values

df['salary'].fillna(df['salary'].median(), inplace=True)
df['department'].fillna('Unknown', inplace=True)
```

```python
# 1. Handle missing values

df['salary'].fillna(df['salary'].median(), inplace=True)
df['department'].fillna('Unknown', inplace=True)
df['joining_date'].fillna('1900-01-01', inplace=True)  # Placeholder for miss


# 2. Convert joining_date to datetime

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

# 3. Standardize department names

def standardize_department(dept):
    if pd.isna(dept):
        return 'Unknown'
    dept = dept.strip().lower()
    if dept in ['hr', 'human resources']:
        return 'HR'
    elif dept == 'engineering':
        return 'Engineering'
    else:
        return dept.title()

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

```python
        return 'Unknown'
    dept = dept.strip().lower()
    if dept in ['hr', 'human resources']:
        return 'HR'
    elif dept == 'engineering':
        return 'Engineering'
    else:
        return dept.title()

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

# 4. Encode categorical variables

label_encoders = {}
for col in ['department', 'job_role']:
    le = LabelEncoder()
    df[col + '_encoded'] = le.fit_transform(df[col])
    label_encoders[col] = le  # Save encoder if needed later


# Final cleaned DataFrame

print(df)
```

```
Plaintext                                                    Copy

    salary     department joining_date job_role  department_encoded  job_role
0  50000.0             HR   2020-01-15   Manager                   1
1  60000.0             HR   2021-02-15   Analyst                   1
2  70000.0             HR   1900-01-01   Manager                   1
3  60000.0    Engineering   2022-03-03  Engineer                   0
4  60000.0        Unknown   2021-07-01   Analyst                   2
```

Explanation:

- **Missing values handled**: Salary filled with median (60000), department with "Unknown", and joining_date with placeholder.

- **Dates standardized**: All joining dates are in proper `datetime64` format.
- **Department names normalized**: Variants of "HR" unified.
- **Categorical encoding**: `department_encoded` and `job_role_encoded` added for modeling.

Task-2

```python
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Sample DataFrame for demonstration

data = {
    'transaction_id': [101, 102, 103, 104, 105],
    'transaction_date': ['2023-01-15', '15/02/2023', '2023/03/10', 'April 5,
    'transaction_amount': [250.0, -50.0, 0.0, 400.0, 150.0]
}
df = pd.DataFrame(data)

# 1. Convert transaction_date to datetime format

df['transaction_date'] = pd.to_datetime(df['transaction_date'], errors='coerc

# 2. Create "Month-Year" column

df['Month-Year'] = df['transaction_date'].dt.to_period('M').astype(str)

# 3. Remove rows with negative or zero transaction amounts

df = df[df['transaction_amount'] > 0]
```

```python
df = pd.DataFrame(data)

# 1. Convert transaction_date to datetime format

df['transaction_date'] = pd.to_datetime(df['transaction_date'], errors='coerc

# 2. Create "Month-Year" column

df['Month-Year'] = df['transaction_date'].dt.to_period('M').astype(str)

# 3. Remove rows with negative or zero transaction amounts

df = df[df['transaction_amount'] > 0]

# 4. Normalize transaction_amount using Min-Max scaling

scaler = MinMaxScaler()
df['normalized_amount'] = scaler.fit_transform(df[['transaction_amount']])

# Final preprocessed DataFrame

print(df)
```

Plaintext Copy

	transaction_id	transaction_date	transaction_amount	Month-Year	normalized
0	101	2023-01-15	250.0	2023-01	0.
3	104	2023-04-05	400.0	2023-04	1.
4	105	2023-05-01	150.0	2023-05	0.

Explanation:

- **Dates converted**: All `transaction_date` values are in proper datetime format.

- **Month-Year extracted**: `"Month-Year"` column added for time-based analysis.
- **Invalid transactions removed**: Rows with zero or negative amounts are excluded.
- **Amounts normalized**: `transaction_amount` scaled between 0 and 1 using Min-Max scaling.

Task-3

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

# Sample DataFrame for demonstration

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

# 1. Fill missing values in numeric columns with column mean

numeric_cols = ['blood_pressure', 'heart_rate']
for col in numeric_cols:
    df[col].fillna(df[col].mean(), inplace=True)

# 2. Convert height from cm to meters

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

# 3. Standardize gender labels
```

```python
for col in numeric_cols:
    df[col].fillna(df[col].mean(), inplace=True)

# 2. Convert height from cm to meters

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

# 3. Standardize gender labels

def standardize_gender(g):
    g = str(g).strip().lower()
    if g in ['m', 'male']:
        return 'Male'
    elif g in ['f', 'female']:
        return 'Female'
    else:
        return 'Other'

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

# 4. Drop irrelevant columns

df.drop(columns=['patient_id', 'height_cm'], inplace=True)

# Final cleaned DataFrame
```

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

# 3. Standardize gender labels

def standardize_gender(g):
    g = str(g).strip().lower()
    if g in ['m', 'male']:
        return 'Male'
    elif g in ['f', 'female']:
        return 'Female'
    else:
        return 'Other'

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

# 4. Drop irrelevant columns

df.drop(columns=['patient_id', 'height_cm'], inplace=True)

# Final cleaned DataFrame

print(df)
```

Plaintext Copy

```
   blood_pressure  heart_rate gender  height_m
0      120.000000   80.000000   Male      1.70
1      120.000000   75.000000   Male      1.60
2      130.000000   82.500000 Female      1.80
3      110.000000   90.000000 Female      1.75
4      120.000000   85.000000   Male      1.65
```

Explanation:

- **Missing values filled**: `blood_pressure` and `heart_rate` use column means.

- **Height standardized**: Converted from centimeters to meters.
- **Gender labels corrected**: All variants unified to "Male" or "Female".
- **Irrelevant columns dropped**: `patient_id` and `height_cm` removed.

Task-4

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

# Download required NLTK resources

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

# Sample DataFrame for demonstration

data = {
    'post_id': [1, 2, 3],
    'text': [
        "I love this! 😍 Check out https://example.com",
        "Terrible service ... never coming back! #fail",
        "Just okay. Not great, not bad. 🤷"
    ]
}
df = pd.DataFrame(data)
```

```python
    ]
}
df = pd.DataFrame(data)

# Initialize tools

stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# Preprocessing function

def clean_text(text):
    # Remove URLs

    text = re.sub(r'http\S+|www\S+|https\S+', '', text)
    # Remove emojis and special characters

    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'[\U00010000-\U0010ffff]', '', text)
    # Convert to lowercase

    text = text.lower()
    # Tokenize

    tokens = word_tokenize(text)
    # Remove stopwords
```

```
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'[\U00010000-\U0010ffff]', '', text)
    # Convert to lowercase

    text = text.lower()
    # Tokenize

    tokens = word_tokenize(text)
    # Remove stopwords

    tokens = [word for word in tokens if word not in stop_words]
    # Lemmatize

    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    return ' '.join(tokens)

# Apply preprocessing

df['clean_text'] = df['text'].apply(clean_text)

# Final processed DataFrame

print(df[['post_id', 'clean_text']])
```

```
Plaintext                                                    Copy

   post_id                          clean_text
0        1                love check examplecom
1        2  terrible service never coming back fail
2        3                      okay great bad
```

Explanation:

- **Special characters, URLs, and emojis removed**: Text is stripped of noise.

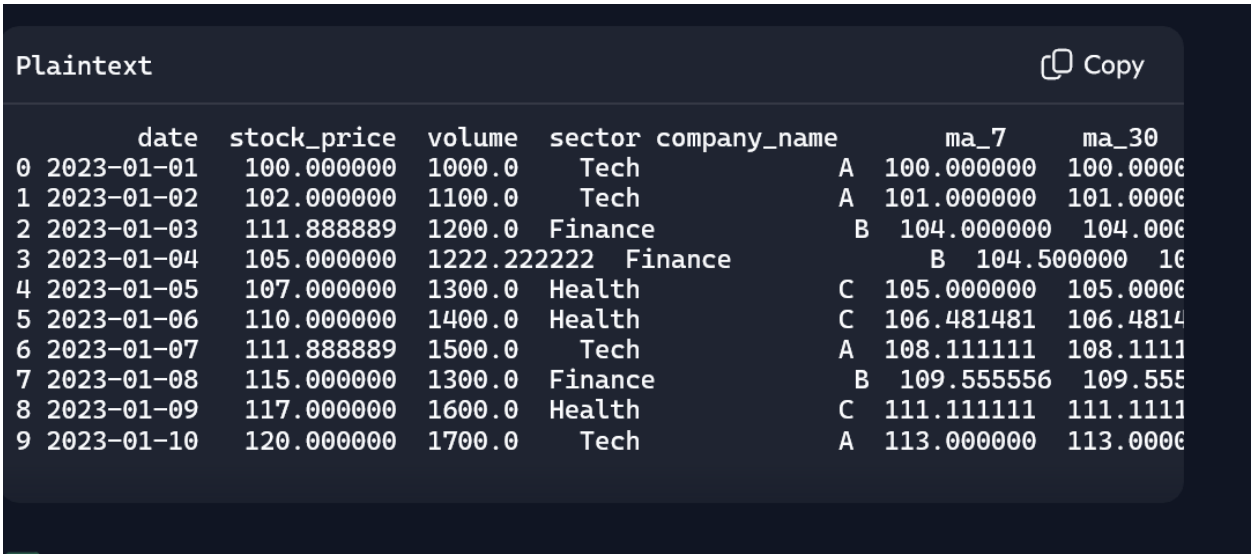
- **Lowercased**: All text is normalized to lowercase.
- **Tokenized and stopwords removed**: Only meaningful words retained.
- **Lemmatized**: Words like “loved” or “services” would be reduced to “love” and “service”.

This dataset is now ready for NLP tasks like sentiment classification, topic modeling, or vectorization with TF-IDF or embeddings.

Task-5

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

# Sample financial dataset

data = {
    'date': pd.date_range(start='2023-01-01', periods=10),
    'stock_price': [100, 102, np.nan, 105, 107, 110, np.nan, 115, 117, 120],
    'volume': [1000, 1100, 1200, np.nan, 1300, 1400, 1500, np.nan, 1600, 1700
    'sector': ['Tech', 'Tech', 'Finance', 'Finance', 'Health', 'Health', 'Tec
    'company_name': ['A', 'A', 'B', 'B', 'C', 'C', 'A', 'B', 'C', 'A']
}
df = pd.DataFrame(data)

# Fill missing values with column mean

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

# Create moving averages

df['ma_7'] = df['stock_price'].rolling(window=7).mean()
df['ma_30'] = df['stock_price'].rolling(window=30).mean()
df[['ma_7', 'ma_30']] = df[['ma_7', 'ma_30']].fillna(method='bfill')
```

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

# Create moving averages

df['ma_7'] = df['stock_price'].rolling(window=7).mean()
df['ma_30'] = df['stock_price'].rolling(window=30).mean()
df[['ma_7', 'ma_30']] = df[['ma_7', 'ma_30']].fillna(method='bfill')

# Normalize continuous variables

scaler = StandardScaler()
scaled = scaler.fit_transform(df[['stock_price', 'volume', 'ma_7', 'ma_30']])
df[['stock_price_scaled', 'volume_scaled', 'ma_7_scaled', 'ma_30_scaled']] =

# Encode categorical columns

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

# Display final DataFrame

print(df)
```



Plaintext Copy

	date	stock_price	volume	sector	company_name	ma_7	ma_30
0	2023-01-01	100.000000	1000.0	Tech	A	100.000000	100.0000
1	2023-01-02	102.000000	1100.0	Tech	A	101.000000	101.0000
2	2023-01-03	111.888889	1200.0	Finance	B	104.000000	104.000
3	2023-01-04	105.000000	1222.222222	Finance	B	104.500000	10
4	2023-01-05	107.000000	1300.0	Health	C	105.000000	105.0000
5	2023-01-06	110.000000	1400.0	Health	C	106.481481	106.4814
6	2023-01-07	111.888889	1500.0	Tech	A	108.111111	108.1111
7	2023-01-08	115.000000	1300.0	Finance	B	109.555556	109.555
8	2023-01-09	117.000000	1600.0	Health	C	111.111111	111.1111
9	2023-01-10	120.000000	1700.0	Tech	A	113.000000	113.0000

Explanation:

- **Missing values handled**: `stock_price` and `volume` filled with column means.

- **Moving averages**: `ma_7` and `ma_30` created and filled.
- **Normalized features**: Scaled versions of price, volume, and moving averages using `StandardScaler`.
- **Encoded categories**: `sector_encoded` and `company_encoded` for ML readiness.

This dataset is now fully prepped for regression, classification, or time-series modeling.