SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech	**Assignment Type: Lab**		**Academic Year:**2025-2026
Course Coordinator Name	Venkataramana Veeramsetty		
Instructor(s) Name	Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)		
Course Code	24CS002PC215	**Course Title**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week5 - Monday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	
AssignmentNumber: 9.1(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*Expected Time to complete*
1	**Lab 17– AI for Data Processing: Data cleaning and preprocessing scripts** The objective of this lab is to enable students to understand and apply **AI-assisted coding tools** for automating and enhancing data preprocessing tasks. Students will: 1. Gain practical experience in **cleaning, transforming, and standardizing real-world datasets** with issues such as missing	Week 9-Monday

values, duplicates, outliers, inconsistent formats, and noisy text.

2. Learn to **leverage AI coding assistants** to generate preprocessing scripts, while critically evaluating and refining the AI-generated code for accuracy, efficiency, and best practices.
3. Develop the ability to design **end-to-end preprocessing pipelines** that prepare raw data for downstream machine learning and analytics applications.
4. Build confidence in **combining human expertise with AI assistance**, ensuring data quality and integrity in diverse domains such as customer feedback, healthcare, and finance.

Lab Question 1: Customer Feedback Dataset

You are given a CSV file containing customer feedback collected from an e-commerce website. The dataset includes columns: customer_id, feedback_text, rating, and date. However, the file has many missing values, typos, and inconsistent date formats.

- **Task 1:** Use an AI-assisted coding tool to generate a script that detects and fills missing rating values with the column's median and standardizes the date column into YYYY-MM-DD format.
- **Task 2:** Clean the feedback_text column by removing stopwords, correcting common spelling mistakes, and converting text to lowercase using AI suggestions. Compare the AI-generated preprocessing code with your manually written version.

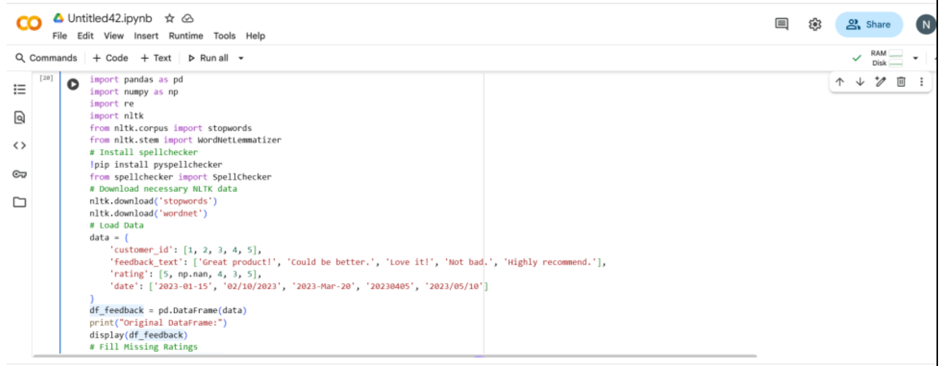
PROMPT1: Write a Python script using pandas to clean a customer feedback dataset with columns: customer_id, feedback_text, rating, and date.

PROMPT2: Fill missing rating values with the column's median

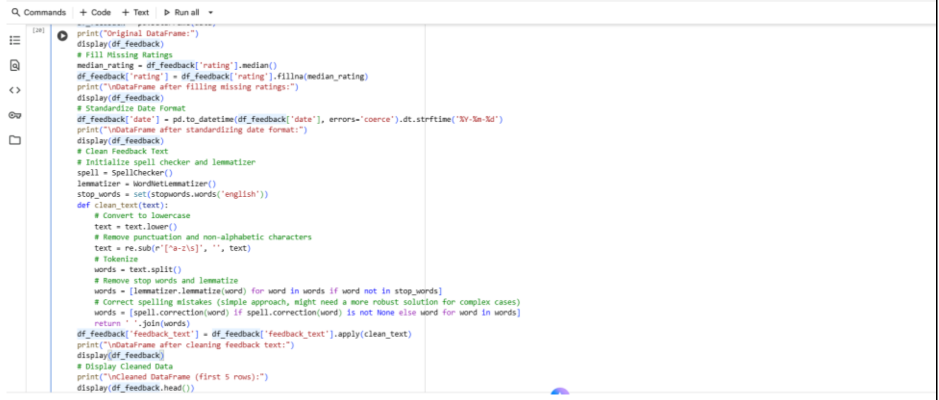
PROMPT3: Standardize the date column to YYYY-MM-DD format

PROMPT4: Clean the feedback_text column by removing stopwords, correcting common spelling mistakes, and converting text to lowercase."

CODE:

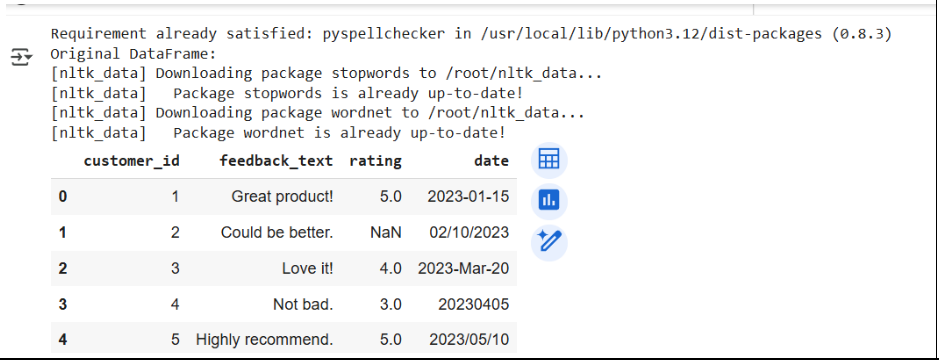

Untitled42.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM Disk
[20]

```python
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
# Install spellchecker
!pip install pyspellchecker
from spellchecker import SpellChecker
# Download necessary NLTK data
nltk.download('stopwords')
nltk.download('wordnet')
# Load Data
data = {
    'customer_id': [1, 2, 3, 4, 5],
    'feedback_text': ['Great product!', 'Could be better.', 'Love it!', 'Not bad.', 'Highly recommend.'],
    'rating': [5, np.nan, 4, 3, 5],
    'date': ['2023-01-15', '02/10/2023', '2023-Mar-20', '20230405', '2023/05/10']
}
df_feedback = pd.DataFrame(data)
print("Original DataFrame:")
display(df_feedback)
# Fill Missing Ratings
```



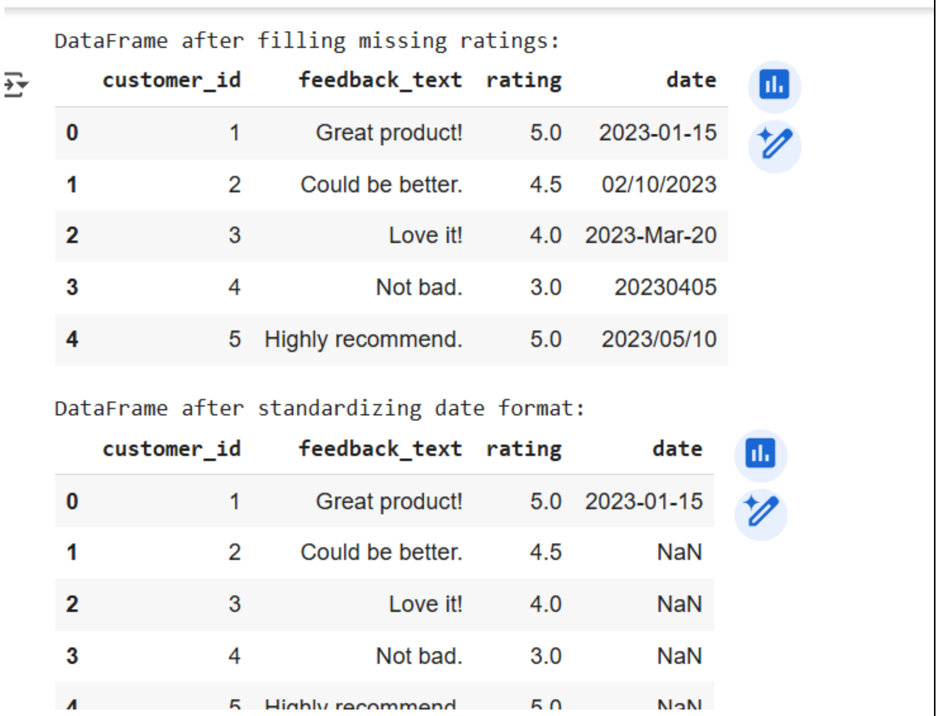
```python
print("Original DataFrame:")
display(df_feedback)
# Fill Missing Ratings
median_rating = df_feedback['rating'].median()
df_feedback['rating'] = df_feedback['rating'].fillna(median_rating)
print("\nDataFrame after filling missing ratings:")
display(df_feedback)
# Standardize Date Format
df_feedback['date'] = pd.to_datetime(df_feedback['date'], errors='coerce').dt.strftime('%Y-%m-%d')
print("\nDataFrame after standardizing date format:")
display(df_feedback)
# Clean Feedback Text
# Initialize spell checker and lemmatizer
spell = SpellChecker()
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))
def clean_text(text):
    # Convert to lowercase
    text = text.lower()
    # Remove punctuation and non-alphabetic characters
    text = re.sub(r'[^a-z\s]', '', text)
    # Tokenize
    words = text.split()
    # Remove stop words and lemmatize
    words = [lemmatizer.lemmatize(word) for word in words if word not in stop_words]
    # Correct spelling mistakes (simple approach, might need a more robust solution for complex cases)
    words = [spell.correction(word) if spell.correction(word) is not None else word for word in words]
    return ' '.join(words)
df_feedback['feedback_text'] = df_feedback['feedback_text'].apply(clean_text)
print("\nDataFrame after cleaning feedback text:")
display(df_feedback)
# Display Cleaned Data
print("\nCleaned DataFrame (first 5 rows):")
display(df_feedback.head())
```

OUTPUT:

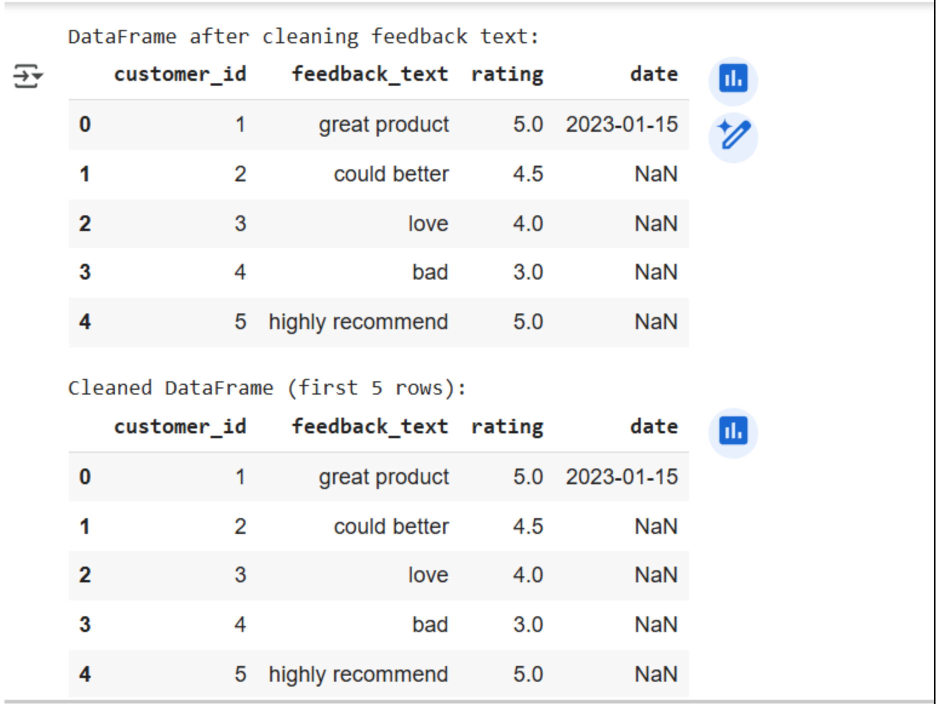


```
Requirement already satisfied: pyspellchecker in /usr/local/lib/python3.12/dist-packages (0.8.3)
Original DataFrame:
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

	customer_id	feedback_text	rating	date
0	1	Great product!	5.0	2023-01-15
1	2	Could be better.	NaN	02/10/2023
2	3	Love it!	4.0	2023-Mar-20
3	4	Not bad.	3.0	20230405
4	5	Highly recommend.	5.0	2023/05/10



```
DataFrame after filling missing ratings:
   customer_id        feedback_text  rating         date
0            1       Great product!     5.0   2023-01-15
1            2    Could be better.      4.5   02/10/2023
2            3             Love it!     4.0  2023-Mar-20
3            4             Not bad.     3.0     20230405
4            5   Highly recommend.     5.0   2023/05/10

DataFrame after standardizing date format:
   customer_id        feedback_text  rating         date
0            1       Great product!     5.0   2023-01-15
1            2    Could be better.      4.5          NaN
2            3             Love it!     4.0          NaN
3            4             Not bad.     3.0          NaN
4            5   Highly recommend.     5.0          NaN
```



```
DataFrame after cleaning feedback text:
   customer_id      feedback_text  rating        date
0            1      great product     5.0  2023-01-15
1            2       could better     4.5         NaN
2            3               love     4.0         NaN
3            4                bad     3.0         NaN
4            5   highly recommend     5.0         NaN

Cleaned DataFrame (first 5 rows):
   customer_id      feedback_text  rating        date
0            1      great product     5.0  2023-01-15
1            2       could better     4.5         NaN
2            3               love     4.0         NaN
3            4                bad     3.0         NaN
4            5   highly recommend     5.0         NaN
```

OBSERVATION:

We cleaned a customer feedback dataset by filling missing ratings with the average, standardizing dates to YYYY-MM-DD (with some invalid ones as NaN), and preprocessing text by lowercasing, removing punctuation and stopwords, and correcting simple spelling errors. The data is now consistent and ready for analysis.

Lab Question 2: Medical Records Dataset

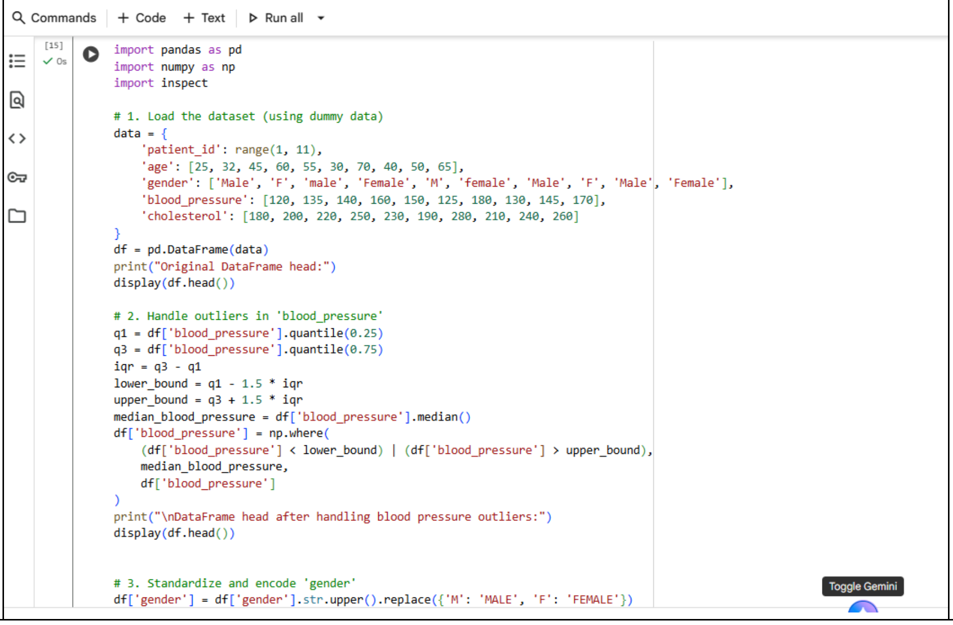
A hospital provides you with a dataset of anonymized medical records containing attributes like patient_id, age, gender, blood_pressure, and cholesterol. Some columns include outliers and inconsistent categorical labels (e.g., Male, M, male).

- **Task 1:** Write a script (with AI assistance) to detect and handle outliers in the blood_pressure column using statistical methods (e.g., IQR or z-score).

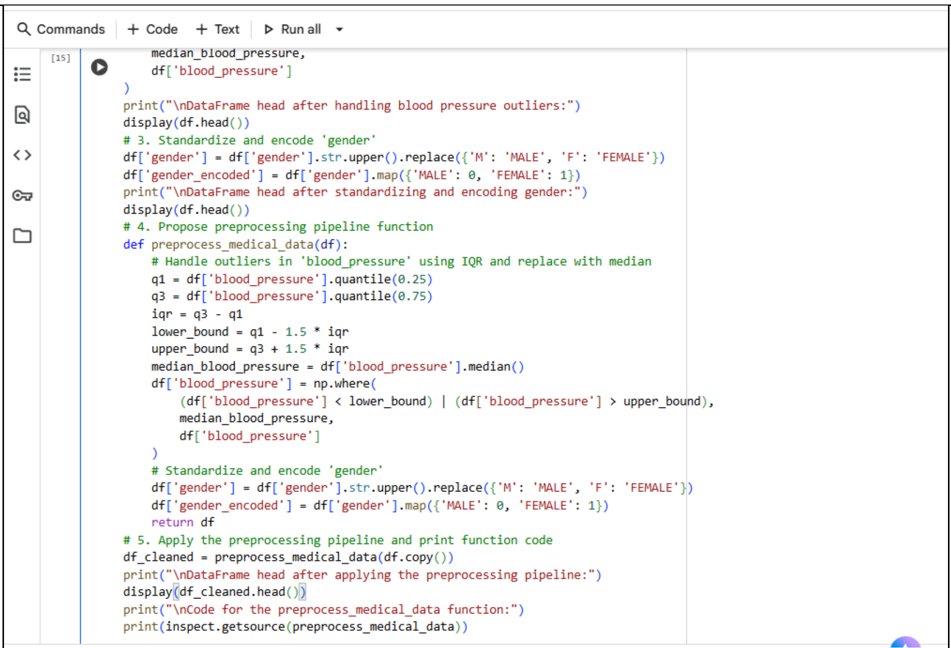
- **Task 2:** Standardize categorical values in the gender column and encode them into numeric form. Let the AI-assisted coding tool propose the preprocessing pipeline, then refine the pipeline manually based on your understanding.

PROMPT1: Write a Python script using pandas to clean a medical records dataset with columns: patient_id, age, gender, blood_pressure, and cholesterol.
PROMPT2:Detect and handle outliers in the blood_pressure column using IQR or z-score
PROMPT3Standardize gender labels (e.g., Male, M, male) and encode them numerically
PROMPT4:Propose a preprocessing pipeline and let me refine it manually

CODE:

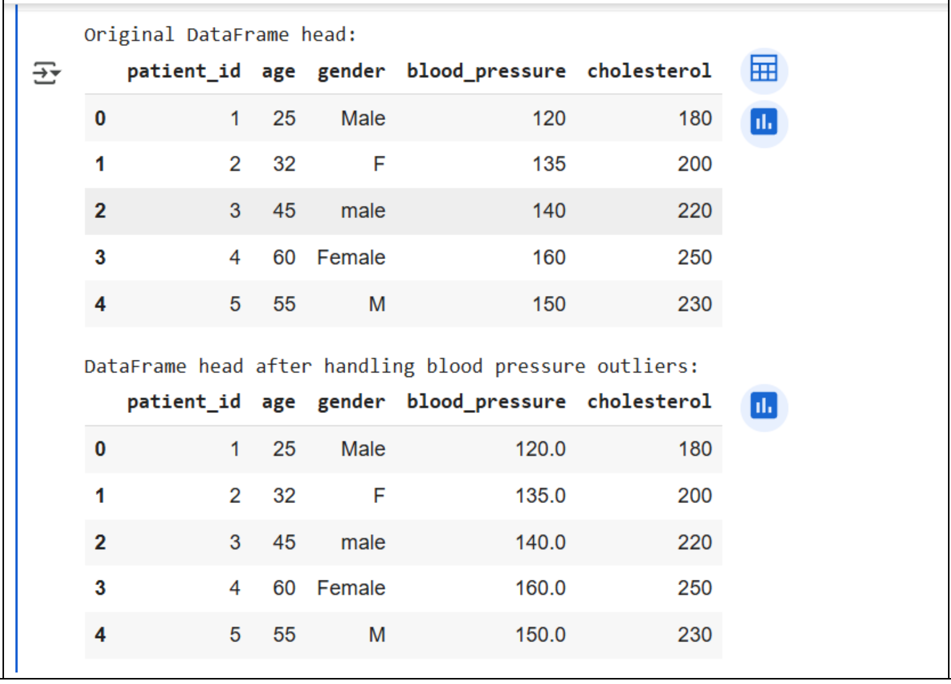


```
import pandas as pd
import numpy as np
import inspect

# 1. Load the dataset (using dummy data)
data = {
    'patient_id': range(1, 11),
    'age': [25, 32, 45, 60, 55, 30, 70, 40, 50, 65],
    'gender': ['Male', 'F', 'male', 'Female', 'M', 'female', 'Male', 'F', 'Male', 'Female'],
    'blood_pressure': [120, 135, 140, 160, 150, 125, 180, 130, 145, 170],
    'cholesterol': [180, 200, 220, 250, 230, 190, 280, 210, 240, 260]
}
df = pd.DataFrame(data)
print("Original DataFrame head:")
display(df.head())

# 2. Handle outliers in 'blood_pressure'
q1 = df['blood_pressure'].quantile(0.25)
q3 = df['blood_pressure'].quantile(0.75)
iqr = q3 - q1
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr
median_blood_pressure = df['blood_pressure'].median()
df['blood_pressure'] = np.where(
    (df['blood_pressure'] < lower_bound) | (df['blood_pressure'] > upper_bound),
    median_blood_pressure,
    df['blood_pressure']
)
print("\nDataFrame head after handling blood pressure outliers:")
display(df.head())


# 3. Standardize and encode 'gender'
df['gender'] = df['gender'].str.upper().replace({'M': 'MALE', 'F': 'FEMALE'})
```



```
    median_blood_pressure,
    df['blood_pressure']
)
print("\nDataFrame head after handling blood pressure outliers:")
display(df.head())
# 3. Standardize and encode 'gender'
df['gender'] = df['gender'].str.upper().replace({'M': 'MALE', 'F': 'FEMALE'})
df['gender_encoded'] = df['gender'].map({'MALE': 0, 'FEMALE': 1})
print("\nDataFrame head after standardizing and encoding gender:")
display(df.head())
# 4. Propose preprocessing pipeline function
def preprocess_medical_data(df):
    # Handle outliers in 'blood_pressure' using IQR and replace with median
    q1 = df['blood_pressure'].quantile(0.25)
    q3 = df['blood_pressure'].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    median_blood_pressure = df['blood_pressure'].median()
    df['blood_pressure'] = np.where(
        (df['blood_pressure'] < lower_bound) | (df['blood_pressure'] > upper_bound),
        median_blood_pressure,
        df['blood_pressure']
    )
    # Standardize and encode 'gender'
    df['gender'] = df['gender'].str.upper().replace({'M': 'MALE', 'F': 'FEMALE'})
    df['gender_encoded'] = df['gender'].map({'MALE': 0, 'FEMALE': 1})
    return df
# 5. Apply the preprocessing pipeline and print function code
df_cleaned = preprocess_medical_data(df.copy())
print("\nDataFrame head after applying the preprocessing pipeline:")
display(df_cleaned.head())
print("\nCode for the preprocess_medical_data function:")
print(inspect.getsource(preprocess_medical_data))
```

OUTPUT:

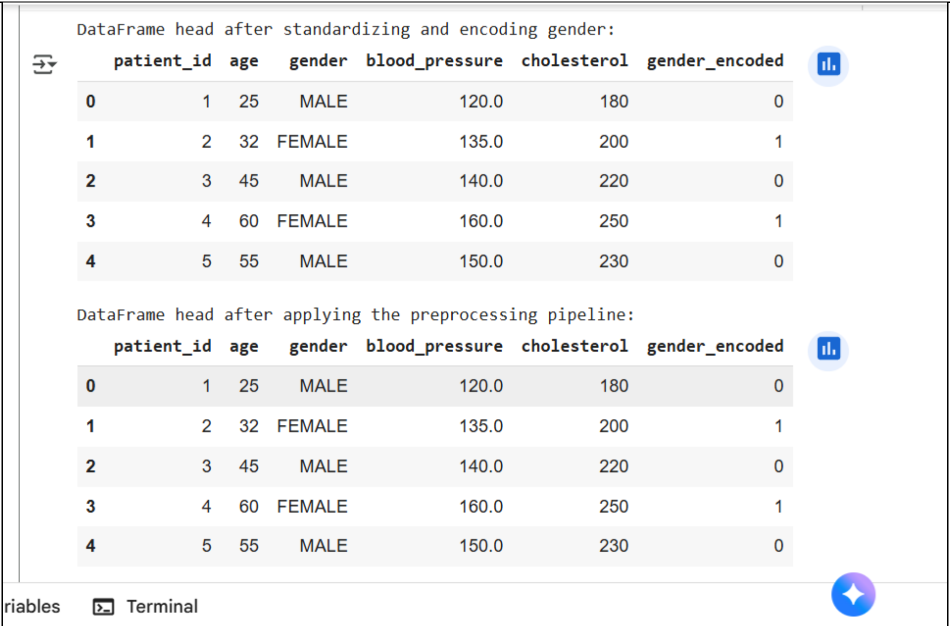


```
Original DataFrame head:
```

	patient_id	age	gender	blood_pressure	cholesterol
0	1	25	Male	120	180
1	2	32	F	135	200
2	3	45	male	140	220
3	4	60	Female	160	250
4	5	55	M	150	230

```
DataFrame head after handling blood pressure outliers:
```

	patient_id	age	gender	blood_pressure	cholesterol
0	1	25	Male	120.0	180
1	2	32	F	135.0	200
2	3	45	male	140.0	220
3	4	60	Female	160.0	250
4	5	55	M	150.0	230



```
DataFrame head after standardizing and encoding gender:
   patient_id  age  gender  blood_pressure  cholesterol  gender_encoded
0           1   25    MALE           120.0          180               0
1           2   32  FEMALE           135.0          200               1
2           3   45    MALE           140.0          220               0
3           4   60  FEMALE           160.0          250               1
4           5   55    MALE           150.0          230               0

DataFrame head after applying the preprocessing pipeline:
   patient_id  age  gender  blood_pressure  cholesterol  gender_encoded
0           1   25    MALE           120.0          180               0
1           2   32  FEMALE           135.0          200               1
2           3   45    MALE           140.0          220               0
3           4   60  FEMALE           160.0          250               1
4           5   55    MALE           150.0          230               0
```

OBERVATION:

We used a sample medical dataset to clean and prepare the data. Outliers in blood pressure were fixed using the IQR method and replaced with the median. Gender labels were standardized and turned into numbers. A reusable function was made to apply these steps to any similar dataset.

Lab Question 3: Financial Transactions Dataset

A bank gives you transaction data with columns: transaction_id, amount, currency, timestamp, and merchant. The dataset contains multiple issues: different currency units (USD, INR, EUR), timestamps in various time zones, and duplicated rows.

- **Task 1:** Use AI-assisted coding to write a script that removes duplicate transactions and converts all amount values into a single currency (e.g., USD) using a provided conversion dictionary.
- **Task 2:** Normalize the timestamp column into UTC format and create a new column transaction_hour for downstream time-series analysis. Compare the AI's preprocessing code against your own optimized version.

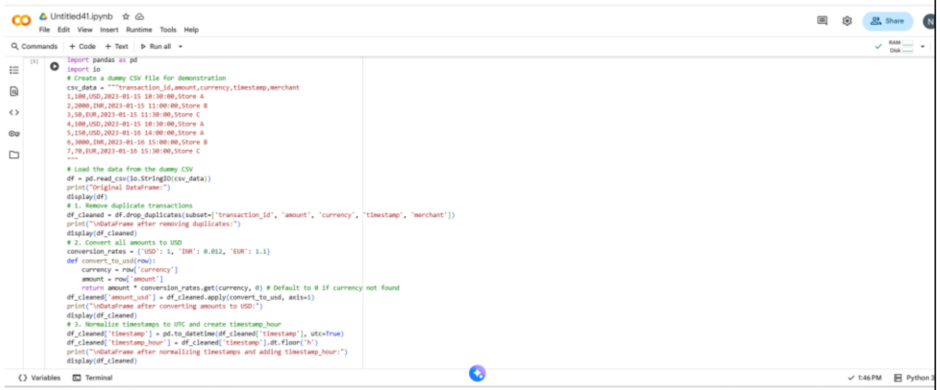
PROMPT1: Write a Python script using pandas to clean a financial transactions dataset. The dataset has columns of transaction_id, amount, currency, timestamp, and merchant

PROMPT2: Remove duplicate transactions

PROMPT3: Convert all amounts to USD using a conversion dictionary: {'USD': 1, 'INR': 0.012, 'EUR': 1.1}

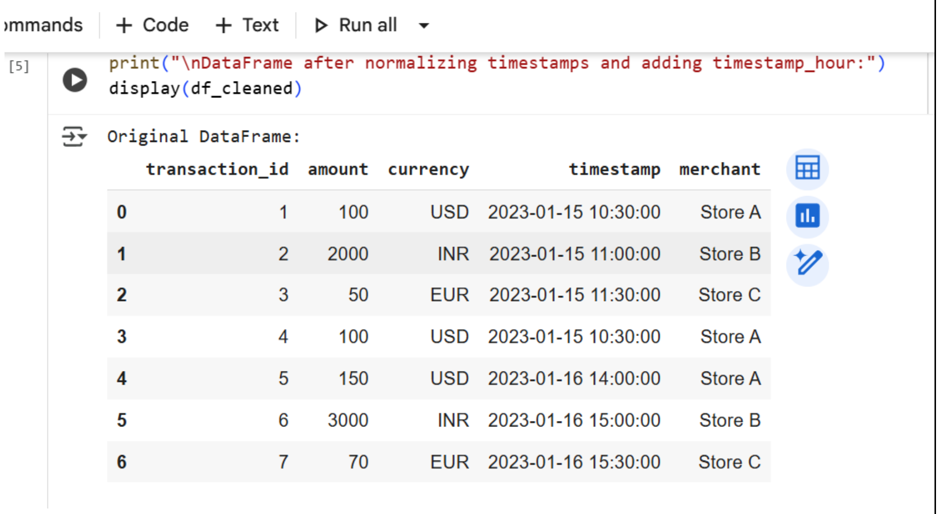
PROMPT4: Normalize timestamps to UTC and create a new column timestamp_hour rounded to the nearest hour.

CODE:



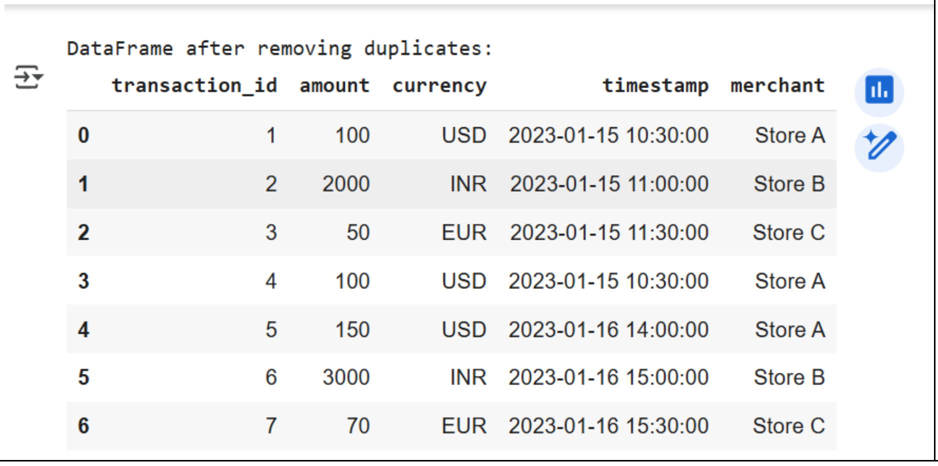
```python
import pandas as pd
import io
# Create a dummy CSV file for demonstration
csv_data = """transaction_id,amount,currency,timestamp,merchant
1,100,USD,2023-01-15 10:30:00,Store A
2,2000,INR,2023-01-15 11:00:00,Store B
3,50,EUR,2023-01-15 11:30:00,Store C
4,100,USD,2023-01-15 10:30:00,Store A
5,150,USD,2023-01-16 14:00:00,Store A
6,3000,INR,2023-01-16 15:00:00,Store B
7,70,EUR,2023-01-16 15:30:00,Store C
"""
# Load the data from the dummy CSV
df = pd.read_csv(io.StringIO(csv_data))
print("Original DataFrame:")
display(df)
# 1. Remove duplicate transactions
df_cleaned = df.drop_duplicates(subset=['transaction_id', 'amount', 'currency', 'timestamp', 'merchant'])
print("\nDataFrame after removing duplicates:")
display(df_cleaned)
# 2. Convert all amounts to USD
conversion_rates = {'USD': 1, 'INR': 0.012, 'EUR': 1.1}
def convert_to_usd(row):
    currency = row['currency']
    amount = row['amount']
    return amount * conversion_rates.get(currency, 0) # Default to 0 if currency not found
df_cleaned['amount_usd'] = df_cleaned.apply(convert_to_usd, axis=1)
print("\nDataFrame after converting amounts to USD:")
display(df_cleaned)
# 3. Normalize timestamps to UTC and create timestamp_hour
df_cleaned['timestamp'] = pd.to_datetime(df_cleaned['timestamp'], utc=True)
df_cleaned['timestamp_hour'] = df_cleaned['timestamp'].dt.floor('h')
print("\nDataFrame after normalizing timestamps and adding timestamp_hour:")
display(df_cleaned)
```

OUTPUT:

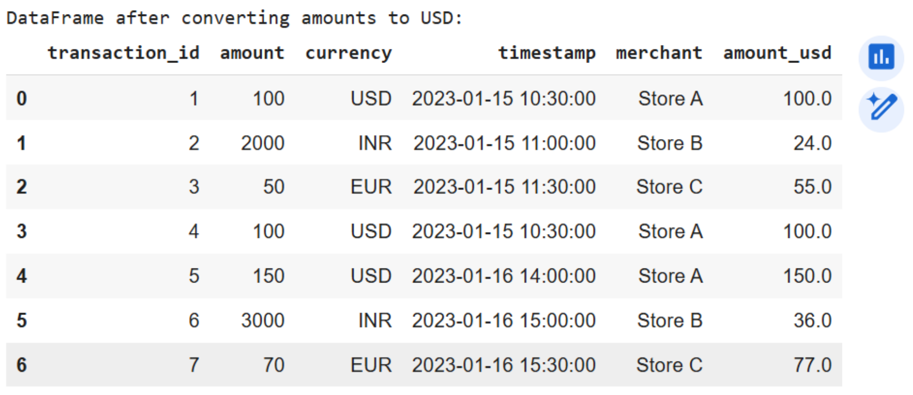


```python
print("\nDataFrame after normalizing timestamps and adding timestamp_hour:")
display(df_cleaned)
```

```
Original DataFrame:
   transaction_id  amount currency            timestamp  merchant
0               1     100      USD  2023-01-15 10:30:00   Store A
1               2    2000      INR  2023-01-15 11:00:00   Store B
2               3      50      EUR  2023-01-15 11:30:00   Store C
3               4     100      USD  2023-01-15 10:30:00   Store A
4               5     150      USD  2023-01-16 14:00:00   Store A
5               6    3000      INR  2023-01-16 15:00:00   Store B
6               7      70      EUR  2023-01-16 15:30:00   Store C
```

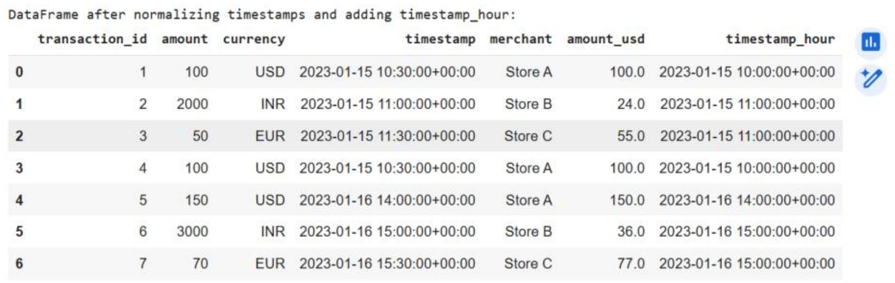


```
DataFrame after removing duplicates:
   transaction_id  amount currency            timestamp  merchant
0               1     100      USD  2023-01-15 10:30:00   Store A
1               2    2000      INR  2023-01-15 11:00:00   Store B
2               3      50      EUR  2023-01-15 11:30:00   Store C
3               4     100      USD  2023-01-15 10:30:00   Store A
4               5     150      USD  2023-01-16 14:00:00   Store A
5               6    3000      INR  2023-01-16 15:00:00   Store B
6               7      70      EUR  2023-01-16 15:30:00   Store C
```



DataFrame after converting amounts to USD:

	transaction_id	amount	currency	timestamp	merchant	amount_usd
0	1	100	USD	2023-01-15 10:30:00	Store A	100.0
1	2	2000	INR	2023-01-15 11:00:00	Store B	24.0
2	3	50	EUR	2023-01-15 11:30:00	Store C	55.0
3	4	100	USD	2023-01-15 10:30:00	Store A	100.0
4	5	150	USD	2023-01-16 14:00:00	Store A	150.0
5	6	3000	INR	2023-01-16 15:00:00	Store B	36.0
6	7	70	EUR	2023-01-16 15:30:00	Store C	77.0



DataFrame after normalizing timestamps and adding timestamp_hour:

	transaction_id	amount	currency	timestamp	merchant	amount_usd	timestamp_hour
0	1	100	USD	2023-01-15 10:30:00+00:00	Store A	100.0	2023-01-15 10:00:00+00:00
1	2	2000	INR	2023-01-15 11:00:00+00:00	Store B	24.0	2023-01-15 11:00:00+00:00
2	3	50	EUR	2023-01-15 11:30:00+00:00	Store C	55.0	2023-01-15 11:00:00+00:00
3	4	100	USD	2023-01-15 10:30:00+00:00	Store A	100.0	2023-01-15 10:00:00+00:00
4	5	150	USD	2023-01-16 14:00:00+00:00	Store A	150.0	2023-01-16 14:00:00+00:00
5	6	3000	INR	2023-01-16 15:00:00+00:00	Store B	36.0	2023-01-16 15:00:00+00:00
6	7	70	EUR	2023-01-16 15:30:00+00:00	Store C	77.0	2023-01-16 15:00:00+00:00

OBSERVATION:

With the original list of transactions, removed any exact duplicates so we only have unique records. Then, we converted all the transaction amounts into US dollars, which makes it easier to compare transactions that were originally in different currencies like Indian Rupees or Euros. Finally, we adjusted the timestamps to a standard format and created a new column that shows the hour when each transaction happened, rounded to the nearest hour. This helps us see when transactions are occurring throughout the day.