LAB ASSIGNMENT

Name: P. Umesh Reddy HallTicket No: 2403a510f9

atch No: 06 Course: AI Assisted Coding

Q1. Stock Price Prediction Setup [5M]

Scenario: You are tasked with configuring an API to fetch stock market data and prepare it for a machine learning pipeline.

• Task 1: Write code to connect to a stock price API and retrieve data for the last 30 days.

Prompt: Write code to connect to a stock price API and retrieve data for the last 30 days.

Code generated:

```
import yfinance as yf
import datetime
import pandas as pd

# Define ticker symbol (Example: Apple Inc.)
ticker_symbol = "AAPL"

# Create Ticker object
ticker = yf.Ticker(ticker_symbol)

# Define date range (last 30 days)
end_date = datetime.date.today()
start_date = end_date - datetime.timedelta(days=30)

# Fetch historical data
historical_data = ticker.history(start=start_date, end=end_date)

# Reset index (Date becomes a column instead of index)
historical_data = historical_data.reset_index()

# Display summary
print(f"📈 Stock Price Data for {ticker_symbol} (Last 30 Days):\n")
print(historical_data[['Date', 'Open', 'High', 'Low', 'Close', 'Volume']])

# Optional: Save to CSV for ML pipeline
historical_data.to_csv(f"{ticker_symbol}_last30days.csv", index=False)
print(f"\n✅ Data saved to {ticker_symbol}_last30days.csv")
```

Output:

```
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html> & C:/Python312/python.exe "c:/Users/THIRUPATHI REDDY/OneDrive/Documents/Desktop/html
/python/task1.py"
$AAPL: possibly delisted; no price data found  (1d 2025-08-02 -> 2025-09-01)
📈 Stock Price Data for AAPL (Last 30 Days):

Empty DataFrame
Columns: [Date, Open, High, Low, Close, Volume]
Index: []

✅ Data saved to AAPL_last30days.csv
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html> 
```

Observations:

API Used: The code uses the yfinance library, which is a reliable and free way to fetch historical stock price data.

Date Range: It correctly calculates the last 30 days using datetime.date.today() and datetime.timedelta(days=30).

Data Retrieval: The ticker.history() method fetches daily OHLCV (Open, High, Low, Close, Volume) data for the specified period.

Data Preparation: The index is reset so that the date becomes a column, making it easier to use in ML pipelines.

Output: The code prints a summary of the key columns and saves the data to a CSV file, which is a common format for ML workflows.

Extensibility: You can easily change the ticker symbol or date range for other stocks or time periods.

Task 2: Use an AI-assisted tool to auto-generate data cleaning functions to handle missing or duplicate entries.

Prompt: Use an AI-assisted tool to auto-generate data cleaning functions to handle missing or duplicate entries.

Code generated:

```python
import yfinance as yf
import datetime
import pandas as pd

# ============================
# Step 1: Fetch stock data
# ============================
def fetch_stock_data(ticker_symbol: str, days: int = 30) -> pd.DataFrame:
    """Fetch last N days of stock data for given ticker."""
    ticker = yf.Ticker(ticker_symbol)
    end_date = datetime.date.today()
    start_date = end_date - datetime.timedelta(days=days)

    historical_data = ticker.history(start=start_date, end=end_date)
    historical_data = historical_data.reset_index()
    return historical_data[['Date', 'Open', 'High', 'Low', 'Close', 'Volume']]

# ============================
# Step 2: Auto-generated cleaning functions
# ============================
def clean_missing_values(df: pd.DataFrame) -> pd.DataFrame:
    """Handle missing values by forward filling, then backward filling."""
    df = df.ffill().bfill()
    return df

def remove_duplicates(df: pd.DataFrame) -> pd.DataFrame:
    """Drop duplicate rows if any exist."""
    df = df.drop_duplicates()
    return df
```

```python
def preprocess_stock_data(df: pd.DataFrame) -> pd.DataFrame:
    """Run all cleaning steps for ML pipeline readiness."""
    df = clean_missing_values(df)
    df = remove_duplicates(df)
    df = df.reset_index(drop=True)
    return df

# ============================
# Step 3: Run pipeline
# ============================
if __name__ == "__main__":
    ticker = "AAPL"
    raw_data = fetch_stock_data(ticker)
    cleaned_data = preprocess_stock_data(raw_data)

    print(f"📊 Cleaned Stock Data for {ticker} (Last 30 Days):\n")
    print(cleaned_data.head())

    # Save for ML pipeline
    cleaned_data.to_csv(f"{ticker}_cleaned_last30days.csv", index=False)
    print(f"\n✅ Data saved to {ticker}_cleaned_last30days.csv")
```

Output:

```
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html> & C:/Python312/python.exe "c:/Users/THIRUPATHI REDDY
/OneDrive/Documents/Desktop/html/python/Labtesttask2.py"
📊 Cleaned Stock Data for AAPL (Last 30 Days):

                       Date        Open        High         Low       Close     Volume
0 2025-08-04 00:00:00-04:00  204.278165  207.644355  201.451371  203.119492   75109300
1 2025-08-05 00:00:00-04:00  203.169408  205.107211  201.930824  202.689957   44155100
2 2025-08-06 00:00:00-04:00  205.396898  215.135845  205.356935  213.008255  108483100
3 2025-08-07 00:00:00-04:00  218.631873  220.599641  216.334477  219.780563   90224800
4 2025-08-08 00:00:00-04:00  220.579666  230.738135  219.001455  229.090012  113854000

✅ Data saved to AAPL_cleaned_last30days.csv
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html>
```

Observations:

Modular Design:

The code is well-structured, separating data fetching, cleaning, and pipeline execution into distinct functions.

Data Fetching:

Uses yfinance to retrieve the last 30 days of stock data for a given ticker. The function is flexible, allowing the number of days to be changed.

Data Cleaning:

Missing Values: Handled using forward fill (ffill) followed by backward fill (bfill), ensuring no gaps remain.

Duplicates: Removed with drop_duplicates().

Index Reset: Ensures a clean, sequential index for ML processing.

Pipeline Readiness:

The preprocess_stock_data function combines all cleaning steps, making the data ready for machine learning workflows.

Output:

Prints a preview of the cleaned data.

Saves the cleaned DataFrame to a CSV file for downstream ML tasks.

Extensibility:

Easy to adapt for other tickers or time ranges by changing function arguments.

Summary:

The script efficiently prepares stock price data for machine learning, ensuring data quality and usability. It follows best practices for data cleaning and modular code organization.

Q2. AI in Healthcare Diagnosis

Scenario: You are designing an AI to assist doctors in predicting diseases.

•Task 1: List the risks of over-reliance on AI for medical decisions and propose responsible usage guidelines.

Prompt: List the risks of over-reliance on AI for medical decisions and propose responsible usage guidelines.

Code generated:

```python
import random
import datetime

# --- Risks Guarded Against ---
# 1. Automation bias -> require human review if confidence low
# 2. Accountability -> audit logs
# 3. Transparency -> return confidence score

# --- Simple AI Diagnostic Function ---
def ai_diagnose(patient_data):
    """
    Dummy AI model that predicts disease risk.
    In real use: plug in a trained ML model.
    """
    # Fake probabilities for "Disease" vs "Healthy"
    prob_disease = random.uniform(0, 1)
    prob_healthy = 1 - prob_disease

    # Decision rule with confidence threshold
    threshold = 0.8
    if prob_disease >= threshold:
        decision = "High risk of disease (doctor confirmation needed)"
    elif prob_disease <= 0.2:
        decision = "Low risk (doctor review recommended)"
    else:
        decision = "Uncertain – mandatory doctor review"
```

```python
    # Audit log (accountability)
    log = {
        "timestamp": datetime.datetime.utcnow().isoformat(),
        "patient_id": patient_data.get("id", "unknown"),
        "prediction": {"disease": round(prob_disease, 2), "healthy": round(prob_healthy, 2)},
        "decision": decision
    }
    print("AUDIT LOG:", log)

    return decision, log

# --- Example Usage ---
patient = {"id": "P001", "age": 45, "symptoms": ["fever", "cough"]}
decision, _ = ai_diagnose(patient)

print(f"\n📋 AI Diagnostic Suggestion: {decision}")
print("⚠️ Reminder: Final decision must be made by a doctor.")
```

Output:

```
AUDIT LOG: {'timestamp': '2025-09-01T10:27:33.678299', 'patient_id': 'P001', 'prediction': {'disease': 0.65, 'healthy': 0.35}, 'decision': 'Uncertain – mandatory doctor review'}

📋 AI Diagnostic Suggestion: Uncertain – mandatory doctor review
⚠️Reminder: Final decision must be made by a doctor.
```

Observations:

Risk Awareness:

The code explicitly addresses key risks: automation bias (requiring human review), accountability (audit logs), and transparency (confidence score).

AI Diagnostic Function:

Uses a dummy model to simulate disease risk prediction.

Generates random probabilities for "disease" and "healthy" outcomes.

Applies a decision rule based on a confidence threshold:

High risk: Doctor confirmation needed.

Low risk: Doctor review recommended.

Uncertain: Mandatory doctor review.

Audit Logging:

Each prediction is logged with a timestamp, patient ID, prediction scores, and decision, supporting traceability and accountability.

Human Oversight:

The final output reminds users that doctors must make the final decision, reinforcing responsible AI usage.

Transparency:

The code prints the audit log, making the AI's reasoning and confidence visible.

Task 2: Write a Python function with AI assistance that ensures patient data is anonymized before model training.

Prompt: Write a Python function with AI assistance that ensures patient data is anonymized before model training.

Code generated:

```python
python > Labtesttask2.py > ...
import pandas as pd
import hashlib

def anonymize_patient_data(df: pd.DataFrame) -> pd.DataFrame:
    """
    Anonymize patient data by removing PII and hashing identifiers if needed.
    """
    pii_columns = ['name', 'id', 'address', 'phone', 'email', 'dob']

    # Remove PII columns
    df = df.drop(columns=[col for col in pii_columns if col in df.columns], errors='ignore')

    # Hash any remaining identifier columns (if present)
    if 'patient_id' in df.columns:
        df['patient_id'] = df['patient_id'].apply(
            lambda x: hashlib.sha256(str(x).encode()).hexdigest()
        )

    # Bin age if present
    if 'age' in df.columns:
        df['age_group'] = pd.cut(
            df['age'],
            bins=[0, 18, 35, 50, 65, 120],
            labels=['0-18', '19-35', '36-50', '51-65', '66+']
        )
        df = df.drop(columns=['age'])

    return df

```

```python
# --------------- Example usage ---------------
patient_df = pd.DataFrame([
    {"patient_id": "P001", "age": 45, "symptoms": "fever, cough", "name": "John Doe"},
    {"patient_id": "P002", "age": 72, "symptoms": "chest pain", "address": "123 Main St"},
])

print("📌 Original Data:")
print(patient_df)

anonymized_df = anonymize_patient_data(patient_df)

print("\n✅ Anonymized Data Ready for Model Training:")
print(anonymized_df)
```

Output:

```
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html> & C:/Python312/python.exe "c:/Users/THIRUPATHI REDDY/OneDrive/Documents/Desktop/html/python/Labtesttask2.py"
📌 Original Data:
  patient_id  age      symptoms      name      address
0       P001   45  fever, cough  John Doe          NaN
1       P002   72    chest pain       NaN  123 Main St

✅ Anonymized Data Ready for Model Training:
                                         patient_id      symptoms age_group
0  df1e40051eff4bcf9b4ebc93083bfcad7f5195746b3e65...  fever, cough     36-50
1  eea502719605f8ec96582d94a0f1738190fe72d16c7b57...    chest pain       66+
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html>
```

Observations:

Purpose:

The function anonymize_patient_data is designed to anonymize patient records before using them for AI model training, which is essential for privacy and compliance.

PII Removal:

It removes common personally identifiable information (PII) columns such as name, id, address, phone, email, and date of birth if they exist in the DataFrame.

Identifier Hashing:

If a patient_id column is present, it hashes the values using SHA-256, making it impossible to reverse-engineer the original IDs.

Age Binning:

If an age column exists, it converts ages into categorical bins (age groups) and removes the raw age value, further protecting privacy.

Return Value:

The function returns a DataFrame with sensitive information removed or masked, suitable for use in AI model training.

Example Usage:

The example demonstrates the anonymization process on a small sample DataFrame and prints both the original and anonymized data for comparison.