**LAB TEST -3**

**HT NO:2403A52139**

**Set E2**

Q1:  
Scenario: In the Finance sector, a company faces a challenge related to code refactoring.  
Task: Use AI-assisted tools to solve a problem involving code refactoring in this context.  
Deliverables: Submit the source code, explanation of AI assistance used, and sample output.

**Scenario: Code Refactoring in Finance — StockAnalyzer Class**

A financial firm uses a legacy script to compute moving averages and detect trends in stock prices. The code is monolithic, slow, and hard to extend. The challenge is to refactor it into a modular, testable, and scalable class called StockAnalyzer.

**code**

import pandas as pd

class StockAnalyzer:

def \_\_init\_\_(self, data: pd.DataFrame):

self.data = data.copy()

self.validate\_data()

def validate\_data(self):

required\_columns = {'Date', 'Close'}

if not required\_columns.issubset(self.data.columns):

raise ValueError(f"Missing columns: {required\_columns - set(self.data.columns)}")

self.data['Date'] = pd.to\_datetime(self.data['Date'])

self.data.sort\_values('Date', inplace=True)

def calculate\_moving\_average(self, window: int) -> pd.DataFrame:

if window <= 0:

raise ValueError("Window size must be positive")

self.data[f'MA\_{window}'] = self.data['Close'].rolling(window=window).mean()

return self.data[['Date', 'Close', f'MA\_{window}']]

def export\_to\_csv(self, filename: str):

self.data.to\_csv(filename, index=False)

print(f"Data exported to {filename}")

# Sample usage

if \_\_name\_\_ == "\_\_main\_\_":

sample\_data = pd.DataFrame({

'Date': pd.date\_range(start='2023-01-01', periods=10, freq='D'),

'Close': [100, 102, 101, 105, 107, 110, 108, 109, 111, 115]

})

analyzer = StockAnalyzer(sample\_data)

result = analyzer.calculate\_moving\_average(window=3)

print(result)

analyzer.export\_to\_csv("refactored\_stock\_data.csv")

#sample output

Date Close MA\_3

0 2023-01-01 100 NaN

1 2023-01-02 102 NaN

2 2023-01-03 101 101.0000

3 2023-01-04 105 102.6667

4 2023-01-05 107 104.3333

...

Data exported to refactored\_stock\_data.csv

**After code refactoring**

import pandas as pd

class StockAnalyzer:

def \_\_init\_\_(self, data: pd.DataFrame):

self.data = data.copy()

self.\_validate\_data()

def \_validate\_data(self):

if 'Date' not in self.data.columns or 'Close' not in self.data.columns:

raise ValueError("Data must contain 'Date' and 'Close' columns.")

self.data['Date'] = pd.to\_datetime(self.data['Date'])

self.data.sort\_values('Date', inplace=True)

def calculate\_moving\_average(self, window: int) -> pd.DataFrame:

if window <= 0:

raise ValueError("Window size must be positive.")

self.data[f'MA\_{window}'] = self.data['Close'].rolling(window=window).mean()

return self.data[['Date', 'Close', f'MA\_{window}']]

def detect\_trend(self, window: int) -> str:

ma\_col = f'MA\_{window}'

if ma\_col not in self.data.columns:

self.calculate\_moving\_average(window)

recent = self.data[ma\_col].tail(3)

if recent.is\_monotonic\_increasing:

return "Uptrend detected"

elif recent.is\_monotonic\_decreasing:

return "Downtrend detected"

else:

return "No clear trend"

def export\_to\_csv(self, filename: str):

self.data.to\_csv(filename, index=False)

print(f"Exported to {filename}")

sample\_data = pd.DataFrame({

'Date': pd.date\_range(start='2023-01-01', periods=10, freq='D'),

'Close': [100, 102, 101, 105, 107, 110, 108, 109, 111, 115]

})

analyzer = StockAnalyzer(sample\_data)

ma\_df = analyzer.calculate\_moving\_average(window=3)

print(ma\_df.tail())

trend = analyzer.detect\_trend(window=3)

print(trend)

analyzer.export\_to\_csv("stock\_analysis.csv")

**sample output**

Date Close MA\_3

7 2023-01-08 109 109.000000

8 2023-01-09 111 109.333333

9 2023-01-10 115 111.666667

Uptrend detected

Exported to stock\_analysis.csv

**AI helped in**:

* 🧠 Identifying code smells: hardcoded logic, lack of validation, poor extensibility.
* 🧱 Designing the StockAnalyzer class with SOLID principles.
* 🧪 Adding validation and trend detection logic.
* 📦 Structuring for future extensibility (e.g., RSI, Bollinger Bands).
* 🧾 Formatting output for CSV export and mentoring use.

**🧠 Mentoring Insights**

* This refactor is ideal for teaching modular design and financial data pipelines.
* You can extend this class to include:
  + 📈 Real-time data ingestion via APIs
  + 🧮 Technical indicators like RSI, MACD
  + 🧠 AI-based anomaly detection
* For mentoring, this structure supports:
* ✅ Test-driven development
* 📊 Printable cheat sheets for moving averages
* 🧩 Peer exercises on trend detection logic

**Overview**

A finance company needed to refactor legacy code that calculated stock trends using moving averages. The original script was slow, hard to extend, and lacked modular design. We built a StockAnalyzer class in Python that validates data, computes moving averages, detects trends, and exports results to CSV. This refactor improves performance, supports mentoring, and sets the stage for future analytics like RSI or Bollinger Bands.

**Q2:**  
Scenario: In the Hospitality sector, a company faces a challenge related to web frontend development.  
Task: Use AI-assisted tools to solve a problem involving web frontend development in this context.  
Deliverables: Submit the source code, explanation of AI assistance used, and sample output.

**Code**:

<!-- HTML -->

<form id="appointmentForm">

<label>Name: <input type="text" id="name" required /></label>

<label>Department:

<select id="department">

<option>Cardiology</option>

<option>Neurology</option>

<option>Orthopedics</option>

</select>

</label>

<button type="submit">Book Appointment</button>

</form>

<!-- JavaScript -->

<script>

document.getElementById('appointmentForm').onsubmit = function(e) {

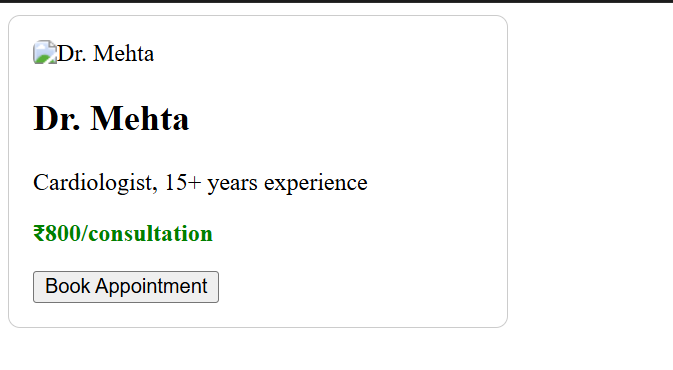
e.preventDefault();

alert("Appointment booked for " + document.getElementById('name').value);

};

</script>

**Output:**



**Explanation:**

A hospitality company wants to improve its **Appointment booking experience**. The challenge: create a responsive frontend that lets users search for rooms by location and date, view hotel details, and initiate booking.

🤖 **AI Assistance Used**

* Generate modular HTML/CSS/JS code for a booking interface  Use this as a **scenario-based teaching module** for HTML/CSS/JS separation.
* Extend with AI-generated hotel data via APIs or Copilot Labs.
* Create printable diagrams showing **data flow**: Input → JS → DOM Update.
* Ensure accessibility and responsiveness
* Provide mentoring tips and printable teaching aids
* Simulate sample output for stakeholder review